Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

Informationstechnik für Luft- und Raumfahrt
Lehrstuhl für Informatik 8 Prof. Dr. Sergio Montenegro

Bachelorarbeit

Implementierung und Evaluierung verschiedener Algorithmen zur autonomen Suche eines Quadrokopters

Vorgelegt von

Paul Barth

Matr.-Nr.: 1766250

Prüfer: Prof. Dr. Sergio Montenegro

Betreuende wissenschaftliche Mitarbeiter: Dipl.-Ing. Nils Gageik

Würzburg, 12. 08. 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit einschließlich aller beigefügter Materialien selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.
Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken entnommen sind, sind in jedem Einzelfall unter Angabe der Quelle deutlich als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.
Mir ist bekannt, dass Zuwiderhandlungen gegen diese Erklärung und bewusste Täuschungen die Benotung der Arbeit mit der Note 5.0 zur Folge haben kann.

Würzburg, 12. 08. 2013

Paul Barth

Aufgabenstellung

Die Fortschritte im Bereich Sensorik und Mikrotechnik ermöglichen heutzutage den kostengünstigen Bau kleiner unbemannter Luftfahrzeuge (UAV, unmanned aerial vehicle, Drohne) wie Quadrokopter. Die Forschung und Entwicklung dieser Systeme wurde in den letzten Jahren aufgrund der vielfältigen Anwendungsmöglichkeiten stark vorangetrieben. Wenngleich im Bereich UAV viel geforscht wurde, ist das Thema Autonomes Flugobjekt längst noch nicht vollständig behandelt. Der Aufbau eines eigenen autonomen Systems wird daher am Lehrstuhl Aerospace Information Technology der Uni Würzburg erforscht und erprobt. Beim vom Universitätsbund geförderten Projekt Lebensretter mit Propellern wird ein System entwickelt, das autonom Räume durchsuchen, Objekte finden sowie deren Position auf einer Karte angeben kann. Im Rahmen dieses Forschungsvorhabens sind geeignete Verfahren zur Koordination der Suche (Suchalgorithmen) zu untersuchen.

Hauptaugenmerk dieser Arbeit ist die Entwicklung eines Algorithmus, der die Suche des Quadrokopters steuert. Denkbar ist die Implementierung üblicher Algorithmen zur Suche wie z.B. Breiten- und Tiefensuche. Die Arbeit soll Vor- und Nachteile unterschiedlicher Suchalgorithmen bezogen auf diesen speziellen Anwendungsfall und unter Berücksichtigung der systemspezifischen Rahmenbedingungen (Sensorik mit Fehlern, Instabiles System & Regelung) behandeln. Die entwickelten Systeme sind in Software per Simulation und am realen System zu evaluieren. Die Entwicklung des realen Systems, das Sensorik und Software zur Positionserfassung, Kollisionsvermeidung und 6DOF Steuer- und Regelung besitzt, ist nicht Teil der Aufgabe. Im Rahmen der Arbeit ist zunächst der Stand der Technik im Bereich autonome Suche aufzuarbeiten und zu beschreiben. Die implementierte Lösung ist in das bestehende System zu integrieren und an diesem ausgiebig zu evaluieren. Die Arbeit ist umfangreich zu dokumentieren.

Aufgabenstellung (Stichpunktartig):

- Aufarbeitung Stand der Technik: Autonome Suche
- Implementierung verschiedener Suchalgorithmen
- Simulation
- Einbettung QT, Integration in Quadrokopter
- Evaluierung am Quadrokopter
- Dokumentation

Zusammenfassung

Mit grundlegenden Elementen aus der allgemeinen Graphensuche, werden in dieser Arbeit mehrere Suchalgorithmen auf Basis eines Quadrokopters implementiert und getestet. Ausgegangen wird von einer Karte bestehend aus Zellen. Diese Zellen können entweder frei oder belegt sein. Die Karte soll komplett abgesucht werden. Für dieses Absuchen werden durch verschiedene Algorithmen Wegpunkte erzeugt. Aufgrund der Komplexität von Karten mit Objekten und Wänden, wurde in dieser Arbeit von einer Karte ausgegangen, bei der keine Hindernisse vorhanden sind. Dafür wurde eine grafische Oberfläche in QT implementiert. In dieser kann ein beliebiger Suchalgorithmus, sowie eine beliebige Startposition ausgewählt werden, wofür Wegpunkte erstellt werden. Weitergegeben werden die Wegpunkte über eine Drahtlosverbindung an einen Quadrokopter. Für drei Suchalgorithmen (Breitensuche, Tiefensuche und ein Zick-Zack-Muster) wurden ausführlich Versuche mit unterschiedlichen Parametern durchgeführt und evaluiert.

Inhaltsverzeichnis

1. Einleitung

Der Schwerpunkt dieser Arbeit liegt darin für einen Raum ohne Hindernisse verschiedene Algorithmen zur Wegpunktgenerierung zu testen. Der Hintergrund dazu ist, sich in einem unbekannten Raum anhand von reinen Onboard-Komponenten zurecht zu finden und diesen Raum systematisch nach Objekten abzusuchen. In einem späteren Stadium soll der Quadrokopter anstelle eines Feuerwehrmannes in ein brennendes Haus fliegen und dort nach Menschen suchen können.
Für das Projekt Lebensretter mit Propellern wird ein kostengünstiges System eines Quadrokopters der Universität Würzburg entwickelt. Dafür werden sowohl in Software als auch Hardware entsprechende Komponenten hinzugefügt. Bisher ist der Quadrokopter in der Lage eine vorgegebene Position autonom anzufliegen, ein oder mehrere Objekte (farbige Kugeln) per Webcam zu erfassen und eine Karte anhand von Ultraschallsensoren zu erstellen. Ebenfalls ist eine Kollisionserkennung bereits implementiert.
Für eine Umgebung in einem brennenden Haus müssen mehrere Komponenten des bisherigen Projektes kombiniert werden. Es wird eine Methode des SLAM (Simultanious Localization And Mapping) benutzt werden, bei der man gleichzeitig eine Karte seiner Umgebung anhand von Sensoren erstellen und seine eigene Position ermitteln wird. Bereits während man diese Karte erstellt, müssen Wegpunkte nacheinander abgeflogen werden um eine vollständige Karte zu erhalten. Auch während dieses Erstellens wird je nach Leistung des Quadrokopters die Objekterkennung parallel laufen können. Das heißt manche Bereiche der Karte werden bereits allein durch das Erstellen der Karte nach Objekten, beziehungsweise Menschen abgesucht sein. Zu beachten ist, dass der Quadrokopter dynamisch eine Kollisionserkennung und -vermeidung umsetzen können werden muss, da ein brennendes Haus keine statische Umgebung sein wird. Die Veränderung von Wänden und Objekten soll dynamisch in der intern gespeicherten Karte aktualisiert werden.
Die Wegpunkte, egal ob zum Erstellen der Karte oder zum Absuchen dieser, sollten sich abhängig von der Kollisionserkennung und der bisher abgesuchten Bereiche (siehe 6.3) dynamisch verän-

dern können. Ebenfalls sollte der Quadrokopter in der Lage sein, bei nicht genauem Erkennen des Suchobjektes, zur selben Stelle zurückzukehren,um eine genauere Betrachtung zu machen (zum Beispiel Flughöhe veringern um ein besseres Bild zu bekommen).
Die komplette Verarbeitung dieser Aufgaben muss in Echtzeit funktionieren. Dafür muss eine gewisse Rechenleistung auf dem Quadrokopter garantiert werden.

2. Stand der Wissenschaft

Suchvorgänge und -algorithmen sind jeden Tag allgegenwärtig: Beginnend im Alltag bei Kreuzworträtseln, dem Rubik's Cube und automatisch einparkenden Autos, bis hin zu Bereichen in Wissenschaft und Forschung wie bei menschenähnlichen Robotern, GPS-navigierenden UAVs (unmanned aerial vehicle, Drohnen), künstliche Intelligenz in Computerspielen, Andockmanöver im Weltraum aber auch in der „computational biology" [LaValle, 2006] zur Darstellung von chemischen Formeln. [LaValle, 2006] Für jedes dieser Gebiete müssen grundlegende Suchkonzepte individuell angepasst und optimiert werden. Einige dieser Suchen werden im Folgenden vorgestellt.

2.1. Grundlagen für eine Suche

Wie wird eine Suche definiert? Was ist eine Suche? Allgemein ist „Suche" ein sehr abstrakter Begriff und wird häufig in der Graphentheorie verwendet. Es gibt Zustände (Knoten), die gewisse Informationen beinhalten um festzustellen, wann ein Zielzustand erreicht ist. Außerdem gibt es Kanten, die den Übergang von einem Knoten zu einem weiteren Knoten beschreiben. Meist geht man von einer baumförmigen Struktur aus, bei der die Suche bei der Wurzel beginnt und über einen oder mehrere der Wege einen Zielzustand erreicht. Es lassen sich viele Probleme zu einer abstrakten Graphstruktur umformulieren. Um Suchverfahren zu vergleichen hat man im Allgemeinen zwei Größen. Es wird einerseits die Zeit (*Time*) bis der Zielknoten erreicht wurde, andererseits der Speicherbedarf (*Space*) für den bisher zurückgelegten Weg zum Zielknoten betrachtet. Weiterhin ist es wichtig, ob ein Suchalgorithmus den Suchraum vollständig absucht und ob eine optimale Lösung, das heißt der vom Startknoten am wenigsten entfernte Zielknoten, gefunden wird. Es müssen jedoch viele Vereinfachungen des Graphen getroffen werden, um eine einheitliche Suchumgebung vorauszusetzen.

1. Sei *L* die Liste der Startknoten für das Problem
   (ab jetzt wird L immer die Liste der noch nicht überprüften Knoten darstellen).
2. Ist *L* leer, so melde einen Fehlschlag.
   Andernfalls wähle einen Knoten *n* aus *L*.
3. Stellt *n* einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten zu *n*.
4. Andernfalls ersetze in *L* den Knoten *n* durch seine Nachfolgeknoten.
   Markiere dabei die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2!

Abbildung 2.2.: Generische Suche [Görz, 2003]

Hier wird von einer Baumstruktur ausgegangen, bei der ein Verzweigungsgrad *b* angibt, wie viele Kinderknoten aus jedem Elternknoten hervorgehen. Jede Ebene von Knoten befindet sich auf einer bestimmten Tiefe *d*, die auch als Abstand vom Wurzelknoten gesehen werden kann. Die Wurzel befindet sich auf Tiefe $d = 0$ (Abbildung 2.1).
Als Grundbaustein für die hier betrachteten Suchverfahren kann man einen leicht erweiterbaren Algorithmus festlegen (Abbildung 2.2). Bei diesem Algorithmus ist der Zustandsraum in Form von Knoten gegeben. Man verwaltet den Weg, das heißt die bisher noch nicht überprüften Knoten, mit einer Liste *L*.

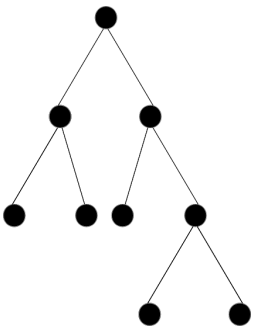


Abbildung 2.1.: Suchbaum mit b=2 und d=3

Neben dem regulären Vorgehen einer Suche vom Start- zum Zielknoten kann man bei genügend Informationen über den Zielzustand und unter der Vorraussetzung, dass die Übergangsoperatoren invertierbar sind, die Suche auch umkehren. Das heißt man vertauscht Start- und Zielknoten und exploriert den Suchraum von der anderen Seite. Einige Suchprobleme lassen sich auf diese Weise einfacher und schneller lösen [Görz, 2003].

2.2. Autonome Suchverfahren

Bei Graphensuchen gibt es zwei verschiedene Gruppen für Suchverfahren, die sich durch die Informationslage bezüglich des Zielzustandes unterscheiden. Bei uninformierten Suchverfahren

sind während des Suchvorgangs keine Informationen über die Lage des Zielzustands vorhanden. Das heißt man muss, um den Zielzustand zu erreichen, den vorhandenen Graphen vollständig systematisch durchsuchen und dabei einen möglichst günstigen Weg vom Wurzelknoten zum Zielknoten ermitteln.
Die informierten Suchverfahren, auch heuristische Suchverfahren genannt, haben bereits von Anfang an die Position des Zielzustands gegeben. Dies ist zum Beispiel der Fall bei einem Parkmanöver. Man weiß genau an welcher Stelle man einparken will, und muss nur noch einen Weg zum Parkplatz finden. Informierte Suchverfahren beschäftigen sich also hauptsächlich damit den geringst möglichen Aufwand, um den Zielknoten zu erreichen, zu ermitteln (Stichwort Kosten). Hier steht also die Pfadfindung (Pathfinding) im Vordergrund [Patnaik, 2006],[Siegwart et al., 2011]).

2.2.1. Uninformierte Suchverfahren

Im Allgemeinen haben uninformierte Suchverfahren einen deutlich größeren Speicherbedarf für die Liste der zu betrachtenden Knoten, als heuristische Suchverfahren, da man hierbei den gesamten Suchraum betrachten muss. Der Zeitbedarf ist dementsprechend bei Weitem größer als der Zeitbedarf einer heuristischen Suche, da eben nicht der kürzeste Weg vom Startknoten sofort gefunden werden kann, sondern zuerst nach dem Ziel "gesucht"werden muss. Hier gilt es stukturiert und sinnvoll den Suchraum nach einem Ziel zu durchforsten.

2.2.1.1. Breitensuche

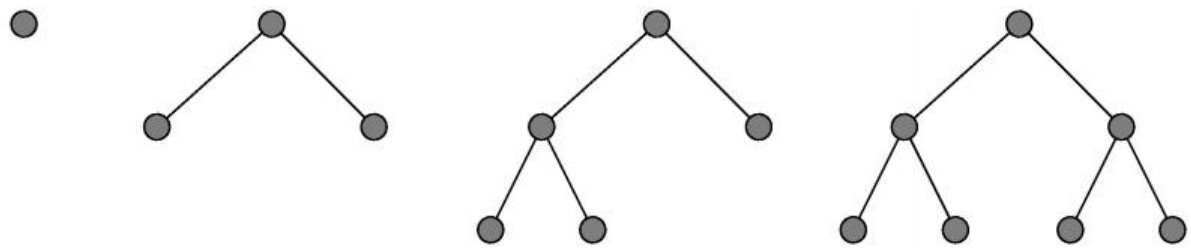


Abbildung 2.3.: Beispiel: Breitensuche [Kleiner und Nebel, 2013]

Die Breitensuche (Breadth-First-Search, kurz BFS) wird dadurch charakterisiert, dass die Suche ebenenweise (also die Breite priorisierend) den Suchgraph absucht (Abbildung 2.3). Dies

1. Sei *L* die Liste der Startknoten für das Problem.
2. Ist *L* leer, so melde einen Fehlschlag.
   Andernfalls wähle den *ersten* Knoten *n* aus *L*.
3. Stellt *n* einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten zu *n*.
4. Andernfalls entferne *n* aus *L* und füge seine Nachfolgeknoten *am Ende* von *L* an.
   Markiere dabei wieder die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2!

Abbildung 2.4.: Breitensuche [Görz, 2003]

geschieht indem man den generischen Suchalgorithmus aus Abbildung 2.2 dahingehend verändert, dass man nicht mehr einen willkürlichen Knoten der Liste betrachtet, sondern den ersten. Weiterhin werden zu dem aktuell betrachteten Knoten die Kinderknoten am Ende der Liste der zu betrachtenden Knoten einfügt. Daraus ergibt sich ein Suchalgorithmus der jeden Ring vom Wurzelknoten aus absucht (Abbildung 2.4). Da die Breitensuche ebenenweise vorgeht, wird, falls es mehrere Zielknoten gibt, derjenige Zielknoten zuerst gefunden, der den geringsten Abstand zum Wurzelknoten hat. Außerdem wird mit einer Breitensuche der Suchraum vollständig abgesucht. Der Speicherbedarf für die Breitensuche ist

$$\mathit{Space}(BFS) = b^d$$

wobei *b* der Verzweigungsgrad und *d* die Tiefe des Suchbaumes sind. Das heißt der Speicherbedarf steigt „ exponentiell in der Tiefe des Suchbaumes!“ [Görz, 2003]
Nachdem man für die Breitensuche ebenenweise den Graph absucht, ergibt sich für die Annahme, dass der Zielknoten in der untersten Ebene des Graphen liegt, ein worst-case Zeitbedarf von:

$$\mathit{Time}(BFS) = O(b^d)$$

[Görz, 2003]

2.2.1.2. Gleiche-Kosten-Suche

Häufig sind die Kanten eines Graphen mit einer Kosten-Variable $c(n \rightarrow n')$ beschriftet, die je nach Aufwand von einem Zustand *n* zum nächsten Zustand *n'* zu wechseln, unterschiedlich groß ist. Die Kosten können sowohl bei uninformierter als auch bei informierter Suche als Unterstützung des Suchverfahrens verwendet werden. Bei heuristischen Suchen können die Kosten gewisse Informationen enthalten, beispielsweise wie weit man vom Zielknoten aktuell entfernt ist. Bei uninformierten Suchen ist zu überlegen, wie man die Kanten sinnvoll beschriftet. Bei der Breitensuche könnte man jede Kante mit 1 beschriften, da die Kosten zur optimalen Lösung damit eindeutig ermittelt werden [Görz, 2003]. Gleiche-Kosten-Suche expandiert vorrangig zuerst diejenigen Knoten, zu denen die Kosten der entsprechenden Kanten möglichst gering sind. Hierbei ist zu beachten, dass Kosten niemals kleiner als eine bestimmte positive untere Schranke ε werden.

$$c(n \rightarrow n') \geq \varepsilon > 0$$

Bei Suchräumen, die keine Bäume sind, können mehrere Pfade zu dem selben Knoten führen (zum Beispiel Zyklen). In diesem Fall wird beim zweiten Betrachten des Knotens entschieden welcher Pfad geringere Kosten vom Startknoten besitzt, und der andere Pfad mit höheren Kosten gelöscht [Görz, 2003].

2.2.1.3. Tiefensuche

Betrachtet man die Breitensuche aus Kapitel 2.2.1.1, ist die Tiefensuche (Depth-First-Search, kurz DFS) quasi das invertierte Gegenstück. Hierbei werden nun nicht mehr Ebene für Ebene die Kinderknoten expandiert, sondern priorisiert in die Tiefe gesucht. Das heißt, der erste Knoten in der Liste *L* wird expandiert, und die Kinderknoten werden direkt an den Anfang der Liste gesetzt. Dies geschieht so lange, bis entweder der Zielknoten gefunden wird, oder der aktuell betrachtete Knoten keine Kinderknoten mehr besitzt. Falls ein Knoten nicht weiter expandierbar ist, geht die Suche so lange schrittweise zurück bis ein bisher nicht betrachteter Kind-Knoten auftaucht (Abbildung 2.5).

Der Speicherbedarf der Tiefensuche ist deutlich kleiner als der der Breitensuche, da lediglich der

1. Sei *L* die Liste der Startknoten für das Problem.
2. Ist *L* leer, so melde einen Fehlschlag.
   Andernfalls wähle den *ersten* Knoten *n* aus *L*.
3. Stellt *n* einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten zu *n*.
4. Andernfalls entferne *n* aus *L* und füge seine Nachfolgeknoten *am Anfang* von *L* an. Markiere dabei wieder die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2!

Abbildung 2.5.: Tiefensuche [Görz, 2003]

Pfad vom Startknoten bis zur Tiefe (maximale Tiefe des Graphen *d*) des Zielknotens gespeichert werden muss.

$$Space(DFS) = O(d)$$

Für den Zeitbedarf des Suchverfahrens, zeigt die Tiefensuche im durchschnittlichen Suchablauf (average-case) keine Veränderung gegenüber der Breitensuche (Berechnung [Görz, 2003]).

$$Time(DFS) = O(b^d)$$

Tiefensuche ist im Gegensatz zur Breitensuche nicht immer vollständig, da der Algorithmus einem unendlich langen Wurzelpfad folgen würde und somit nie den Zielknoten erreichen kann. Außerdem findet die Tiefensuche auch nur bedingt den optimalen Zielknoten. Da zuerst in die Tiefe die Kinderknoten betrachtet werden, würde bei mehreren Zielknoten immer der Zielknoten links unterhalb des optimalen Knoten zuerst geliefert werden [Görz, 2003].

2.2.1.4. Schrittweise vertiefende Suche

Da die Tiefensuche einen deutlich kleineren Speicherbedarf und in günstigen Suchräumen auch einen besseren Zeitbedarf als die Breitensuche aufweist, hat man einen Algorithmus entwickelt, der von beiden Suchalgorithmen die positiven Aspekte kombiniert. Die schrittweise vertiefende Suche (iterative deepening, kurz ID) führt im Wesentlichen eine ebenenweise Tiefensuche durch (Abbildung 2.6). Dies garantiert das Finden der optimalen Lösung und gleichzeitig auch

1. Sei $c = 1$ (c steht für die maximale Suchetiefe).
2. Sei L die Liste der Startknoten für das Problem.
3. Ist L leer, so erhöhe c um 1 und
   mache weiter mit Schritt 2!
   Andernfalls sei n *der erste* Knoten in L.
4. Stellt n einen Zielknoten dar,
   so melde Erfolg und liefere den Pfad vom Startknoten zu n.
5. Andernfalls entferne n aus L.
   Befand sich n auf einer Tiefe kleiner als c,
   so füge *an den Anfang* von L die Nachkommen von n an.
   Markiere dabei die neuen Knoten jeweils mit dem zugehörigen Wurzelpfad, der beim Startknoten beginnt.
6. Weiter mit Schritt 3!

Abbildung 2.6.: schrittweise vertiefende Suche [Görz, 2003]

die Vollständigkeit des Suchverfahrens mit dem Speicherbedarf der Tiefensuche und einen maximal dreifachen Zeitbedarf der Tiefensuche [Görz, 2003].

2.2.2. Heuristische Suchverfahren

Bei uninformierten Suchen ist der Zeitbedarf immer $O(b^d)$ [Hopcroft und Ullman, 1979], solange die Suche „Anspruch auf Vollständigkeit erhebt“[Görz, 2003]. Das heißt die Zeit steigt exponentiell abhängig von der Tiefe des Suchraumes. Um diesen Zeitbedarf deutlich zu verringern benutzt man Informationen über den zu findenden Zielzustand, beziehungsweise Informationen über die Suchumgebung. Man benötigt eine heuristische Schätzfunktion h, welche eine Entscheidung für einen Nachfolgeknoten herbeiführt, der näher am Zielknoten ist als andere Knoten.
Im Folgenden liegt der Fokus der Suchalgorithmen darauf, möglichst schnell und effizient einen Zielzustand zu erreichen. Hierbei kann jedoch nicht immer garantiert werden, dass der Aufwand wirklich minimal ist. In manchen Fällen verfolgt der Algorithmus seinen Pfad in eine Sackgasse und findet keinen Zielzustand [Lunze, 1994].

1. Sei L die Liste der Startknoten für das Problem, *sortiert* nach der jeweils geschätzten Distanz h zum Ziel.
2. Ist L leer, so melde einen Fehlschlag.
   Andernfalls wähle den *ersten* Knoten n aus L.
3. Stellt n einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten zu n.
4. Andernfalls entferne n aus L und füge seine Nachfolgeknoten, sortiert nach der jeweils geschätzten h Distanz zum Ziel, *am Anfang* von L an.
   Markiere dabei die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2!

Abbildung 2.7.: Das Bergsteigerverfahren [Görz, 2003]

2.2.2.1. Bergsteigen

Das Bergsteigerverfahren (hill climbing with backtracking, kurz BTHC) nutzt die heuristische Schätzfunktion um die expandierten Knoten zu sortieren. Beim Suchvorgang wird dann der Knoten mit dem besten Wert an den Anfang der Liste gesetzt und demnach auch als nächstes betrachtet (Abbildung 2.7). „Läßt man die heuristische Schätzfunktion jeden Knoten mit 0 bewerten, so degeneriert die Bergsteigersuche zur reinen Tiefensuche.“[Görz, 2003] Vollständigkeit kann in unendlichen Suchräumen vom Bergsteigerverfahren nicht gewährleistet werden [Görz, 2003].

2.2.2.2. Optimistisches Bergsteigen

Im Vergleich zum Bergsteigerverfahren, wird beim optimistischen Bergsteigen (strict hill climbing, kurz SHC) nicht jeder Kindknoten in die Liste L aufgenommen, sondern nur der Beste (Abbildung 2.8). Dies senkt den Speicherbedarf des Verfahrens deutlich im Vergleich zum Bergsteigerverfahren. Dadurch, dass nicht alle Kinderknoten betrachtet werden, können einige Komplikationen im Suchvorgang auftreten. Beispielsweise kann sich der Algorithmus in einem lokalen Minimum, welches nicht zwingend ein Zielknoten sein muss, verrennen und dort nie wieder herausfinden. Der Suchalgorithmus findet auch nur extrem langsam einen Zielknoten, wenn mehrere oder alle Kinderknoten die gleiche heuristische Distanz haben (Plateus). Es gibt einige Ideen,

1. Sei *L* die Liste der Startknoten für das Problem, *sortiert* nach der jeweils geschätzten Distanz *h* zum Ziel.
2. Ist *L* leer, so melde einen Fehlschlag.
   Andernfalls wähle den *ersten* Knoten *n* aus *L*.
3. Stellt *n* einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten zu *n*.
4. Andernfalls entferne *n* aus *L* und füge (nur) den besten Nachfolgeknoten von *n* (den mit minimaler geschätzter Distanz *h* zum Ziel) *am Anfang* von *L* an.
   Markiere dabei den neuen Knoten mit dem zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2!

Abbildung 2.8.: Optimistisches Bergsteigen [Görz, 2003]

um diese Probleme zu behandeln. Eine Variante beginnt, um nicht in einem lokalen Minimum gefangen zu sein, die Suche an einer *zufälligen* Stelle im Graphen wieder neu [Görz, 2003].

$$Space(SHC) = O(b)$$

2.2.2.3. Gierige Suche

Die gierige Suche (greedy search, kurz GS) betrachtet nicht wie das Bergsteigerverfahren nur die Nachfolgeknoten mit der heuristischen Schätzfunktion, sondern erweitert den Bereich auf den gesamten Graphen (Abbildung 2.9). Das heißt die Nachfolgeknoten werden danach ausgewählt, ob sie *geschätzt* am nähesten zum Zielknoten sind. Allgemein ist die Gierige Suche nicht vollständig. Für endliche Suchräume ist sie jedoch vollständig. Der Speicherbedarf der Gierigen Suche ist deutlich größer als der des Bergsteigerverfahrens [Görz, 2003].

2.2.2.4. A*-Suche

Während die Gierige Suche und das Bergsteigerverfahren Schwierigkeiten mit lokalen Minima hat, zieht der A* Algorithmus den kompletten Suchraum in Betracht und versucht einen möglichst besten, also oben liegenden Knoten zu finden. Die heuristische Schätzfunktion wählt den Nachfolgeknoten abhängig von der Tiefe des Zielknotens und gleichzeitig den Knoten, welcher schätzungsgemäß am nähesten zum Zielknoten liegt. In Abbildung 2.10 wird die Ausführung des

1. Sei L die Liste der Startknoten für das Problem.
2. Ist L leer, so melde einen Fehlschlag.
   Andernfalls wähle denjenigen Knoten n aus L, der dem Ziel *schätzungsgemäß am nächsten* ist.
3. Stellt n einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten zu n.
4. Andernfalls entferne n aus L und füge alle seine Nachfolgeknoten in die Liste L ein. Markiere dabei wieder die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten.
5. Weiter mit Schritt 2!

Abbildung 2.9.: Gierige Suche [Görz, 2003]

A* Algorithmus gezeigt. Für genauere Informationen zur Schätzfunktion und dem A* Algorithmus siehe [Görz, 2003], [Hart et al., 1968] und [Lunze, 1994].

2.2.3. Chaosdrive

Roboter die im häuslichen Gebrauch nützliche Aufgaben übernehmen, wie zum Beispiel den Rasen mähen, oder Staubsaugen, wird die Aufgabe der Suche mit Hilfe von Zufall deutlich vereinfacht. Das Prinzip ist so stupide wie einfach: Der Roboter fährt geraudeaus, bis er eine Wand oder künstliche Begrenzung trifft, dreht sich in einem zufälligen Winkel von der Wand weg und fährt dann weiter. [Sommer, 2008] Mit dieser Vorgehensweise kann nur auf eine unendlich lange Zeit garantiert werden, dass der komplette Bereich abgesucht wurde. Durch den Faktor Zufall kann man auch statistisch keine Versuche mit Chaosdrive auswerten. Man müsste sich damit zufriedengeben, dass der Roboter nur einen gewissen Teil des Suchgebiets besucht hat.

2.3. UAV Suchen

2.3.1. Flugzeugähnliche UAVs

Aufgrund der physikalischen Eigenschaften von flugzeugähnlichen UAVs müssen Suchverfahren möglichst lange Strecken geradeaus abfliegen. Als Muster zum Absuchen von Gebieten bietet sich ein zeilenweises Abscannen der Umgebung an. Am Ende jeder Zeile wird dann eine 180 °

1. Initialisiere die Agenda L
   mit dem Startknoten s für das Problem: L:=$\{s\}$.
2. Initialisiere die Liste bereits erschlossener Knoten C mit $\emptyset$.
3. Ist L leer, so melde einen Fehlschlag.
4. Andernfalls wähle denjenigen Knoten n aus L, für den

   $$f(n) = g(n) + h(n)$$

   minimal ist (dabei bezeichne $g(n)$) die Kosten des günstigsten Pfads vom Startknoten s zu n, der bis jetzt gefunden wurde),
   entferne n aus L und trage n in C ein.
5. Stellt n einen Zielknoten dar, so melde Erfolg und liefere den Pfad vom Startknoten s zu n.
6. Andernfalls füge alle Nachfolgeknoten von n, die nicht schon in C enthalten sind, in die Liste L ein.
   Markiere dabei die neuen Knoten mit dem jeweils zugehörigen Pfad vom Startknoten s.
   Sollte einer der in L aufzunehmenden Nachfolgeknoten c von n bereits in L enthalten sein, so mache keine neue Kopie, sondern *ändere die Markierung* von c auf den kürzesten Pfad vom Startknoten s, der bis jetzt gefunden wurde.
7. Weiter mit Schritt 3!

Abbildung 2.10.: A*-Suche [Görz, 2003]

Drehung (U-Turn) durchgeführt. Dieser U-Turn sollte außerhalb des Suchbereichs liegen, da die Kamera bei einer so scharfen Drehung nicht mehr exakt auf den Boden gerichtet wird [Bishop, 2010],[Coleman et al., 2012].

2.3.2. Helikopterähnliche UAVs

Für helikopterähnliche UAVs gibt es zwei unterschiedliche Anwendungsgebiete. Im Gegensatz zu flugzeugähnlichen UAVs können Helikopter auch im Indoor-Bereich arbeiten. Dies bringt sowohl Vor- als auch Nachteile mit sich. Man ist im Inneren eines Gebäudes nicht mehr in der Lage die Position über GPS zu lokalisieren. Man benötigt also die Möglichkeit seine eigene Position festzustellen. Dafür bietet sich SLAM (Simultanous Localization And Mapping) an. Um sich in einer Umgebung zurecht zu finden kann man eine Kamera benutzen und mithilfe der Änderung des Kamerabildes feststellen in welche Richtung man sich bewegt [Soundararaj et al., 2009].

Weiterhin werden für Abstandsmessungen der Umgebung bei einigen Projekten Laser-Sensoren verwendet [Grzonka et al., 2009]. Man muss innerhalb von Gebäuden, vor allem beim autonomen Fliegen, den Fokus viel mehr auf Kollisionserkennung legen, als im Outdoor-Bereich.

2.4. Suchen mit mehreren Robotern

In [Sarid und Shapiro, 2009] wird ausführlich eine Vorgehensweise für das Absuchen einer Umgebung mit Wänden und Objekten mit mehreren bodenständigen Robotern erklärt. Dabei wird die Umgebung vom Startpunkt aus erst in den umgebendenden Kreis und dann von dort in weitere Ringe nach außen hin unterteilt. Jedem Roboter wird ein Bereich zum Absuchen zugeteilt. Der Roboter nutzt eine Kollisionserkennung um nicht mit Wänden oder anderen Robotern zu kollidieren. Die einzelnen Gebiete werden per Tiefensuche vom Roboter abgesucht und dann per Kommunikation mit den anderen Robotern ein weiter außen liegender, noch nicht zugewiesener Ring erforscht. Für Quadrokopter gibt es die Möglichkeit des Formationsflugs, wobei die Quadrokopter ebenfalls untereinander kommunizieren. Somit wäre eine Suche mit mehreren Quadrokoptern in einem ähnlichen Schema durchaus denkbar.

3. Konzept

3.1. Überblick

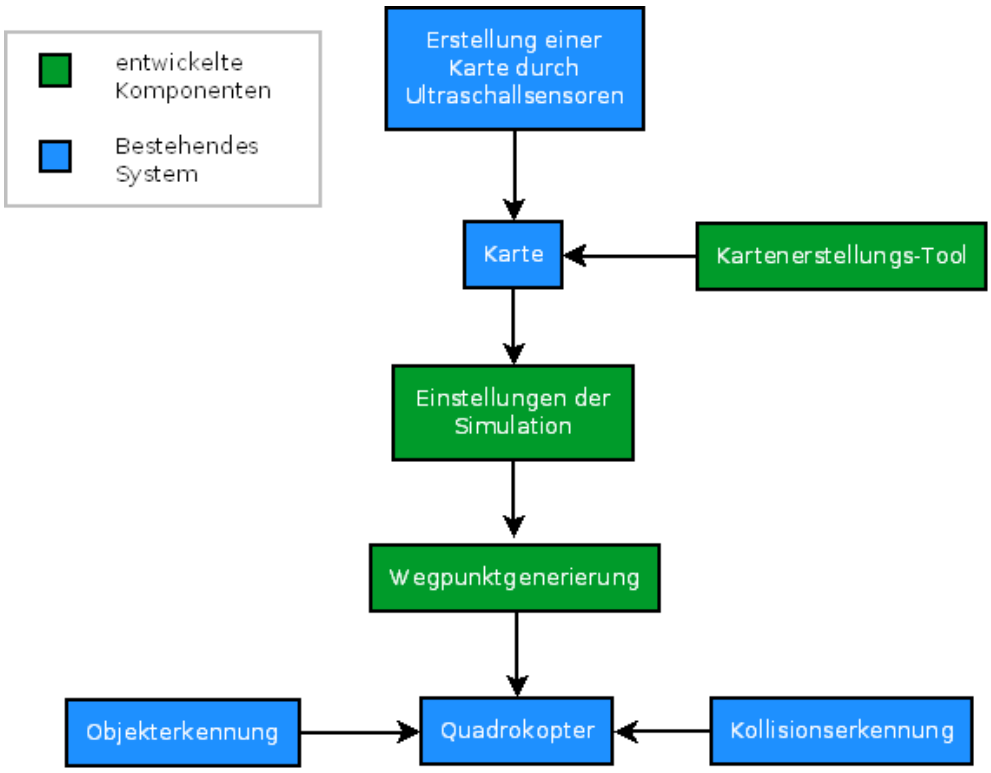


Abbildung 3.1.: Gesamtkonzept

Das Ziel dieser Arbeit ist, mithilfe einer übergebenen Karte, für den Quadrokopter Wegpunkte zu erstellen, welche diese Karte vollständig absuchen. Um eine Karte zu erstellen gibt es neben dem Mapping-Verfahren, welches in der Arbeit „Intelligentes Mapping für Indoor-Quadrokopter“ von [Schmitt, 2012] angewandt wird die Möglichkeit, die Karte per GUI (Graphical User Interface) selbst zu “zeichnen“. Die zu erzeugenden Wegpunkte sind davon abhängig welcher Algorithmus verwendet werden soll, um die Karte abzusuchen. Zusätzlich ist die Startposition je nach Algorithmus ein wichtiger Parameter. Da die Priorität auf einem uninformierten Suchprinzip liegt,

wurde die Komplexität der Karte auf ein Minimum herunter gesetzt. Die Karte zum Generieren der Wegpunkte ist komplett leer, bis auf eine Begrenzung am Rand. Neben dem Vorhaben, in der übergebenen Karte Objekte zu finden, muss auch die Kollisionserkennung in komplexeren Umgebungen beachtet werden, da der Quadrokopter ein sehr instabiles Flugmodel hat (siehe Kapitel 5.3 und Kapitel 6.2).

3.2. Iteratives Suchen

Meist wird in der Robotik bei Suchproblemen von bodenständigen Robotern ausgegangen, welche problemlos schrittweise (iterativ) das Vorgehen im Suchverlauf berechnen können. Dies ist ein herausragender Vorteil gegenüber fliegender Roboter (UAVs, zum Beispiel Quadrokopter), da diese bereits beim Auf-der-Stelle-Fliegen deutliche Abweichungen zur gewünschten Position haben. Dies rührt daher, dass die Sensorik der Positionsbestimmung durch Vibrationen veränderte Messwerte erhalten, und somit eine Bewegung berechnet wird. Außerdem ist der Quadrokopter ein instabiles System, da die Regelung der Motoren abhängig von der verbauten Hardware jeweils neu an das veränderte Gewicht optimiert werden muss. Ein bodenständiger Roboter kann im Allgemeinen davon ausgehen, dass seine Koordinaten sich nicht verändern, während er steht. Bei einer Suche dieses Roboters ist es folglich nicht wichtig, wie lange die Berechnung der nächsten Position dauert. Beim Anfliegen eines Punktes, kippt der Quadrokopter, was eine veränderte Messung des optischen Flusssensors zur Folge hat. Laut dieser Messung denkt der Quadrokopter ein kleines Stück zurück geflogen zu sein, jedoch hat sich seine Position nicht verändert, sondern er wurde nur gekippt. Wenn der Quadrokopter mehrere einzelne Wegpunkte auf einer geraden Strecke anfliegt, werden viele dieser Messfehler erhalten, folglich versucht man möglichst wenige Wegpunkte zu erstellen.

3.3. Wegpunkt-basiertes Suchen

Anhand der in Kapitel 3.2 beschriebenen Problematik muss für das Quadrokopter-Projekt nach einer besseren Vorgehensweise gesucht werden. Es wurde durch empirische Versuche festgestellt, dass der Quadrokopter auf einer langen geraden Strecke Schwierigkeiten hat, die einzelnen Zwischenpunkte anzufliegen. Deshalb liegt die Überlegung nahe, dem Quadrokopter nur den letzten

Wegpunkt der Strecke zu übergeben. Der Schwerpunkt dieser Arbeit liegt also darin, diese Wegpunkte möglichst sinnvoll zu wählen.

3.3.1. Problematik der Ausführung

Dadurch, dass man nicht permanent seine Umgebung für den nächsten Schritt betrachten kann, ergeben sich sehr viele Fälle, die nicht trivial zu lösen sind. Deutlich wird dies an einem einfachen Beispiel:
Der Quadrokopter will von einem zum nächsten Wegpunkt fliegen. Jedoch liegt der nächste Wegpunkt in einer Wand. Was soll er tun?
Dies ist bei weitem nicht der einzige Fall, bei dem eine Entscheidung gefällt werden muss. Je nach Suchalgorithmus, entstehen auch Probleme wenn eine freie Zelle der Karte komplett von Wänden oder Objekten umgeben ist. Der Algorithmus würde dann versuchen diesen Punkt zu erreichen, obwohl er unnerreichbar ist.

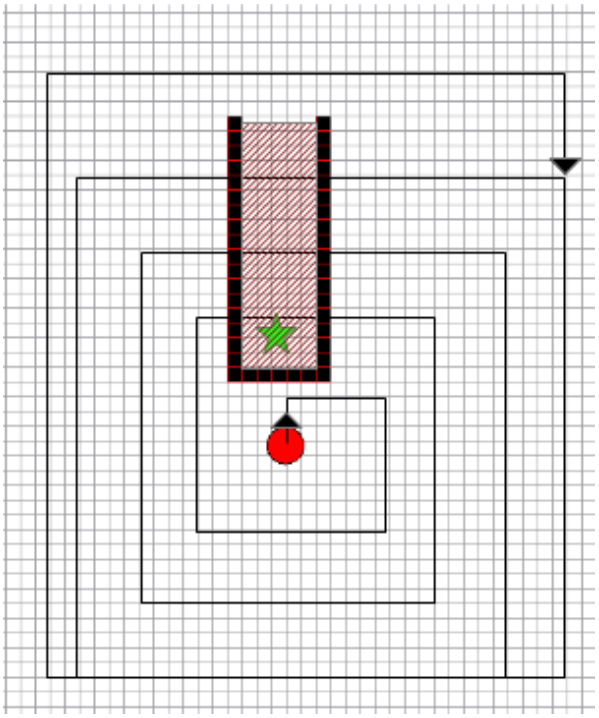


Abbildung 3.2.: Ziel im Korridor nicht erreichbar

Wenn sich ein Zielzustand am Ende eines langen Korridors befindet, dessen Eingang jedoch vom jetzigen Zustand abgewandt ist, muss der Suchalgorithmus diesen finden, wenn die Wegpunkte am Eingang vorbeiführen würden. Der rot schraffierte Bereich in Abbildung 3.2 kann per Wegpunktgenerierung nicht erreicht werden. Der Quadrokopter würde per Pathfinding um die Wände herum fahren und keine Wegpunkte beim Ziel (hier: grüner Stern) erstellen.

Bei einem simplen zeilenweisen Absuchen eines Raumes, der Wände und/oder Objekte enthält, kann allein schon darin die Schwierigkeit bestehen, den Startpunkt des Absuchens festzulegen. Dies wäre eine heuristische Suche zu einer beliebigen Ecke im Raum. In ungünstigen Fällen, ist der Aufwand den Startpunkt zu finden bereits höher, als der Aufwand den Zielzustand zu erreichen! In Abbildung 3.3 wäre zum Beispiel ein zeilenweises Absuchen des Raumes mit Aus-

gangspunkt in der linken unteren Ecke nicht vollständig, beziehungsweise der Aufwand in die linke unterste Ecke zu kommen bereits selbst ein Suchproblem.

3.3.2. Lösungsansätze

Da diese Arbeit im Wesentlichen Breiten- und Tiefensuche behandelt, werden im Folgenden primär für diese Verfahren Ansätze für die Probleme aus Kapitel 3.3.1 besprochen.

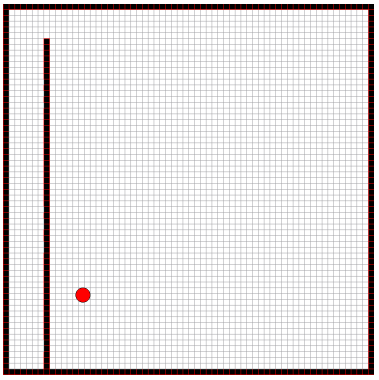


Abbildung 3.3.: Ungünstiger Fall für Zeilenweises Suchen

Breitensuche:

Um die Zellen von Innen nach Außen zu betrachten, ohne eine Zelle zu vergessen, wird in den Zellen zusätzlich zur Wie-Oft-Besucht-Variable noch eine Ring-Variable gespeichert. Diese Ring-Variable wird beim Start der Simulation mit einem Wert von $0 \ldots \infty$ belegt, wobei der Wert angibt wie weit die Zelle vom Startpunkt entfernt ist. (Den Wert ∞ sollte also keine Ring-Variable einer Zelle erreichen, solange die Karte nicht unendlich groß ist.) Wenn der Suchalgorithmus startet, werden die Wegpunkte so gelegt, dass zuerst alle Zellen des niedrigsten Rings abgesucht werden, soweit möglich. Wenn die Suche auf einem höheren Ring ist und eine Zelle mit niedrigerem Ring passiert, wird der Algorithmus in die Zelle mit niedrigerem Ring fortgesetzt. Zu beachten ist, dass belegte Zellen (Wände und Ähnliches) mit keiner Ring-Variable belegt werden müssen, da diese sowieso nicht betrachtet werden. Diese Ringvariable wird bei der Entscheidungsfällung bei einer Wand auf zwei verschiedene Arten verwendet.

- *Priorität Innen:* Beim Vorfinden einer Wand geht der Algorithmus priorisiert auf einen niedrigeren Ring, auch wenn dieser bereits betrachtet worden ist, und versucht dadurch einen Weg zum nachfolgenden Wegpunkt zu finden.

- *Priorität Außen:* Trifft der Algorithmus auf eine Wand verfolgt er solange seinen bisherigen Pfad zurück, bis der Weg zum nächst höheren Ring frei ist. Dies sorgt dafür, dass bereits betrachtete Zellen nicht unnötig oft betrachtet werden. Beachte: Der Algorithmus geht hierbei trotzdem auf einen niedrigeren Ring zurück, sobald eine noch nicht betrachtete Zelle mit niedrigerem Ring betrachtet wird.

Tiefensuche:

Für die Tiefensuche wurden zwei Varianten entworfen, die jedoch teilweise Wegpunkte nah beieinander erstellen.

- Man hält an dem einfachen Schema fest: „Laufe geradeaus bis eine Wand kommt oder du bereits dort warst, biege dann rechts ab!“ Unter Betrachtung von Wänden oder Objekten im Raum ergibt sich das Problem, dass durch das Schema manche Bereiche gar nicht betrachtet werden. Deshalb müsste man sich entweder beim Abbiegen merken, ob auf der anderen Seite (hier: in Flugrichtung links) ein bisher unbesuchter Bereich liegt. Diesen muss man sich dann merken und *später* wieder anfliegen.

- Alternativ könnte man das Schema etwas abändern: „Laufe geradeaus bis eine Wand kommt oder du bereits dort warst, biege dann rechts ab! Betrachte permanent die Umgebung links von dir und biege links ab, sobald der Bereich frei und unbesucht ist.“ Dieses Schema würde dazu führen, dass man einen Raum von außen nach innen absucht, und dabei immer kleinere Ringe zieht.

Da die Problematik unter Betrachtung der Wände eher Lösungsansätze im Bereich Pathfinding benötigt, wurde im Rahmen dieser Arbeit davon ausgegangen Wegpunkte zum Absuchen eines leeren Raumes zu generieren.

4. Implementierung

4.1. Überblick

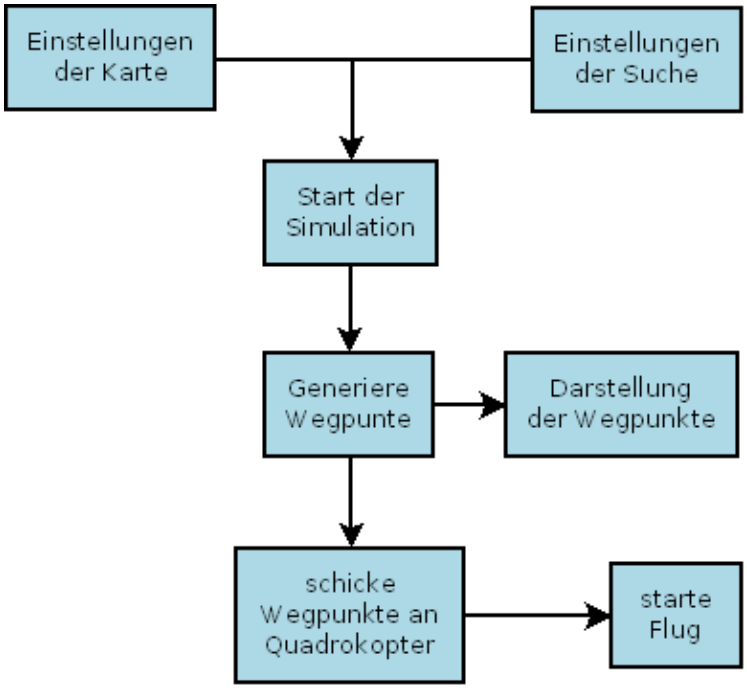


Abbildung 4.1.: Software

Wie man in Abbildung 4.1 sehen kann, werden für die Generierung der Wegpunkte zuerst die Einstellungen der Karte (Zellgröße, Kartengröße, Wände) und die Einstellungen der Suche (Startposition, Suchalgorithmus) fixiert. Durch diese Einstellungen können die Wegpunkte für die meisten Algorithmen direkt erstellt werden (Ausnahme: Chaosdrive oder Ähnliche, da keine Abbruchbedingung). Die Darstellung der Wegpunkte bezeichnet die Umsetzung der Grafik im GUI, welche entweder die Wegpunkte direkt einzeichnet, oder in einer Simulation diese schrittweise anfährt. Nachdem die Wegpunkte erstellt worden sind, müssen diese zuerst an den Quadrokopter geschickt werden. Daraufhin muss ein Start-Kommando zum Fliegen gesendet werden.

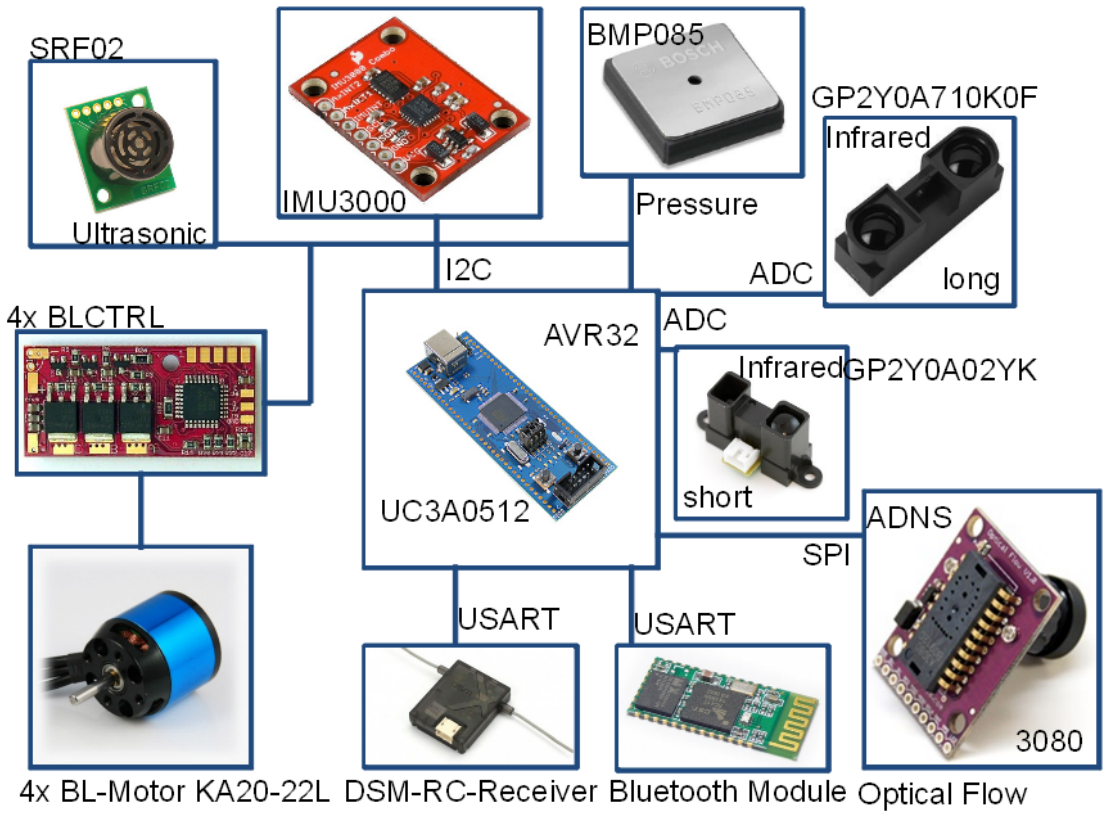


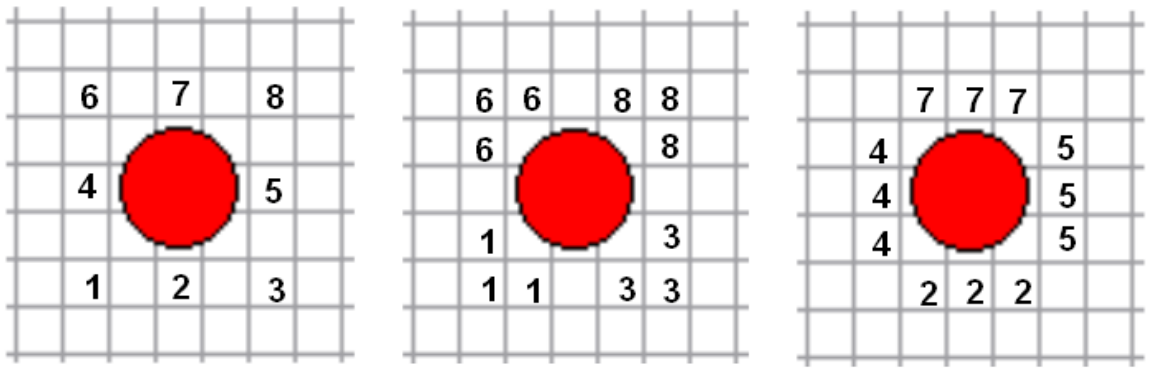
Abbildung 4.2.: Bestehendes System

Im Rahmen dieser Arbeit wurde an der Hardware Struktur nahezu nichts verändert. Zusätzlich zu der in Abbildung 4.2 dargestellten Komponenten wurde ein Infrarot-Sender am Quadrokopter befestigt. Dieser dient dazu die Position des Quadrokopters mit Hilfe der vier Kameras des optischen Tracking-Systems aufzunehmen.
Zum Abfliegen der Wegpunkte nutzt der Quadrokopter seine intern bekannte Position. Diese wird durch den optischen Flusssensor berechnet [Strohmeier, 2012].

4.2. Anpassung der Suchverfahren

Allgemein geht man bei einer autonomen Suche von einem Graphen (zum Beispiel in Baumform) aus. Bei dem vorliegenden Fall muss man das Suchverfahren anpassen. Die Umgebung besteht aus einer zweidimensionalen Karte, welche Zellen von bestimmter Breite und Höhe beinhaltet. Die Breite und Höhe des Quadrokopters ist von Anfang an vorgegeben. Das heißt der Quadroko-

pter belegt bereits von selbst eine gewisse Anzahl von Zellen, abhängig davon welche Größe die Zellen haben. Um einen Suchvorgang zu betreiben, wurde folgende Überlegung angestellt: Bei der Suche betrachtet man immer die direkt anliegenden Zellen in acht Richtungen und summiert den Wert, wie oft eine Zelle bereits besucht worden ist, auf eine jeweilige Variable auf. Mit dieser Vorgehensweise kann getestet werden, in welcher Richtung der Quadrokopter sich bisher am wenigsten aufgehalten hat. Die Umsetzung dieser Überlegungen ist nur teilweise im Endergebnis übrig geblieben. Einige Suchverfahren konnten in der Praxis nicht getestet werden. (siehe dazu Kapitel 4.3.2)
In Abbildung 4.3 wird davon ausgegangen, dass der Quadrokopter (hier: roter Punkt) nicht direkt auf dem Raster der Karte liegt. Wenn dieser direkt auf einer Rasterung liegt, würden andere Zellen aufsummiert werden. Man kann diesem Problem vorbeugen, indem man den Startpunkt des Quadrokopters direkt auf ein festes Raster legt. Diese feste Rasterung wurde nicht implementiert, da das Aufsummieren der besuchten Zellen keinen in der Praxis getesteten Suchalgorithmus betrifft, sondern lediglich zur Darstellung der betrachteten Zellen in der Simulation genutzt wurde.



(a) Richtungen (b) Ecken (c) Kanten

Abbildung 4.3.: Zählvariablen

4.3. GUI

Das hier implementierte **G**raphical **U**ser **I**nterface dient dazu einen von verschiedenen Suchalgorithmen auszuwählen und für den Quadrokopter eine Liste von Wegpunkten zu generieren. Dabei kann man verschiedene Einstellungen vornehmen (Abbildung 4.4):

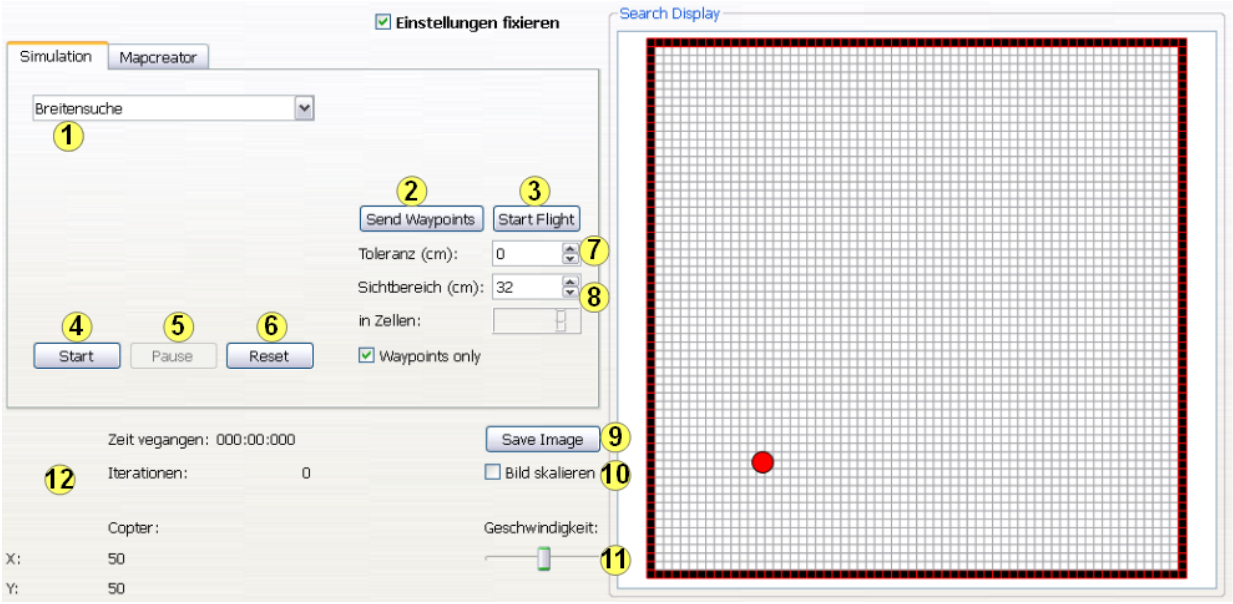


Abbildung 4.4.: Simulations-Tab

1:

Combobox zur Auswahl der unterschiedlichen Algorithmen. Wird während der Simulation deaktiviert

2:

Button zum Schicken der generierten Wegpunkte an den Quadrokopter. Die Liste der Wegpunkte wird erst am Ende der Simulation generiert und so lange intern gespeichert, bis erneut auf den Start-Button (**4**) gedrückt wird. Das heißt die selben Wegpunkte können auch nach dem Resetten des Quadrokopters erneut geschickt werden, ohne diese neu generieren zu müssen.

3:

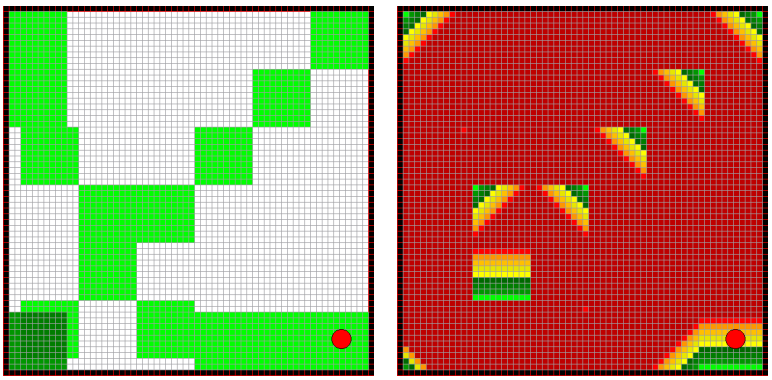
Schickt dem Quadrokopter das Kommando den Flug zu starten. Vorher müssen die Wegpunkte an den Quadrokopter geschickt werden (siehe **2**)

4:

Deaktiviert alle Eingabeoptionen, aktiviert den Pause-Button (**5**). Genereriert die Wegpunkte für den ausgewählten Algorithmus (**1**) und bewegt den Quadrokopter (roter Punkt), je nach Einstellung der Waypoints-Only-Checkbox, direkt zu den Wegpunkten oder zeichnet die Zwischenschritte ein (siehe Abbildung 4.5).

5:

Unterbricht die Simulation der Wegpunktgenerierung und erlaubt einen anderen Suchalgorithmus (**1**) auszuwählen. Die ursprüngliche Idee dahinter war, dass man verschiedene Suchalgo-



(a) Eingeschaltet (b) Ausgeschaltet

Abbildung 4.5.: Waypoints only-Checkbox

rithmen hintereinander einsetzen kann um komplexe Umgebungen unterschiedlich abzusuchen. Nachdem die ausgewählten Algorithmen zum Generieren der Wegpunkte aber nur für leere Räume ausgelegt sind, wird dies nicht weiter benötigt.

6:

Setzt die Grafik auf den Ausgangszustand zurück. Erlaubt wieder alle Einstellungen zu verändern.

7:

Die zu generierenden Wegpunkte werden um diesen Wert "herausgeschoben", das heißt der Quadrokopter muss jeweils ein Stück weiter fliegen um den Wegpunkt zu erreichen. Da der Quadrokopter bereits einige Zentimeter vor dem Wegpunkt ein Erreichen des Wegpunktes vermerkt, kann mit dieser Variable das Anfliegen der Wegpunkte variiert beziehungsweise optimiert werden. Genaueres siehe Kapitel 5.2.

8:

Der Sichtbereich kann abhängig von den Kameraeinstellungen zur Erkennung von Objekten variiert werden. Der abzudeckende Bereich der Kamera hängt vom Öffnungswinkel der Kamera und der Flughöhe ab. Der hier einzustellende Paramter verändert die Wegpunktgenerierung so, dass die Wegpunkte für einen kleinen Sichtbereich näher beieinander liegen, et vice versa.

9:

Speichert die aktuelle Grafikanzeige in einer Bilddatei (Dateitypen: jpg, png oder bmp).

10:

Wenn die Checkbox aktiviert ist, kann man per Mausrad die Grafik skalieren. Zusätzlich kann man per Mausklick die Grafik verschieben, zum Beispiel wenn man zu nah heran gezoomt hat.

11:
Wird zum Einstellen der Simulationsgeschwindigkeit genutzt. Intern wird eine Zeitvariable zum Verzögern verwendet, die in einem Bereich von zwei Sekunden bis eine Millisekunde skaliert wird. Diese Einstellung beeinflusst **nicht** die Geschwindigkeit des Quadrokopters in der Realität zum Anfliegen der Wegpunkte, sondern bestimmt lediglich die Geschwindigkeit der Simulation.

12:
Anzeige der aktuellen Koordinaten des Simulations-Quadrokopters, der vergangenen Zeit zum Generieren der Wegpunkte und die Anzahl der gebrauchten Schritte zum bisherigen Wegpunkt. Die vergangene Zeit wird erst aktualisiert wenn auf Pause (**5**) oder Reset (**6**) gedrückt wurde.

4.3.1. Benutzte Suchalgorithmen

Aufgrund der Komplexität der Suche in einer nicht trivialen Umgebung, wurden hier die Suchalgorithmen für eine komplett freie Umgebung gewählt (siehe Kapitel 3.3.1). Diese Algorithmen dienen dazu einen Gegenstand in einem, durch die eingstellte Karte beschränkten, Raum zu suchen. Es wird davon ausgegangen, dass sich keine Hindernisse in Form von Wänden oder Objekten in dem Raum befinden. Die Karte beschränkt die Algorithmen in dem Sinne, dass per Voreinstellung nur bestimmte Kartengrößen eingegeben werden können, da jede Teil-Karte mindestens 64 Zellen haben muss. Man könnte manuell beliebig kleinere Wände einzeichnen, das wäre jedoch zu aufwendig. Die Umsetzung der Suchalgorithmen ist in zwei Teile unterteilt. Erst werden die Wegpunkte anhand der Position des Startpunktes und der Belegung der Zellen generiert. Im zweiten Schritt werden diese generierten Wegpunkte in der Grafik abgefahren. Das heißt die Generierung der Wegpunkte kann auch komplett von der Grafik unabhängig geschehen. Ausnahme für diese Vorgehensweise sind die zwei Chaosdrive Varianten, da diese ihre Wegpunkte erst bei der Bewegung in der Simulation speichern.

Breitensuche:

Eine Breitensuche bedeutet in dieser Versuchsanordnung, dass man zuerst seine nächsten Nachbarn betrachtet und dann ebenenweise immer weiter voranschreitet. Dies wurde so umgesetzt, dass man jeweils um eine Länge der eingestellten Sichtweite weiter nach außen vorgeht. Wenn der Toleranz-Parameter größer als 0 ist wird der Wegpunkt um diesen Bereich weiter nach außen geschoben, so lange der Wegpunkt nicht in einer Wand liegt. Die

Wegpunkte werden spiralförmig im Uhzeigersinn generiert. Beim Erreichen einer Begrenzung wird nicht die Drehrichtung geändert, sondern die bereits vorher betrachteten Bereiche nochmals überflogen. Da der Quadrokopter ein instabiles System ist, fällt es ihm schwer Wegpunkte, die nahe beieinander liegen, abzufliegen. Durch einen Richtungswechsel der Drehrichtung müsste jedoch ein Wegpunkt direkt neben dem aktuellen erstellt werden, obwohl der Quadrokopter noch den Schwung der langen Strecke vorher hat (Abbildung 4.6). Die folgende Abbruchbedingung wurde für die Generierung der Wegpunkte entworfen: Es wird jeweils der kleinste und größte X- und Y-Wert gespeichert. Aus diesen Berechnet man die horizontale und vertikale Distanz vom jeweils kleinsten und größten Wert. Wenn sich die Distanz in der X- und Y-Richtung nach mehrmaligem Betrachten nicht verändert, wird die Generierung abgebrochen und die Wegpunkte aus der Liste gelöscht, welche zu viel generiert wurden. Die grafische Abarbeitung der Wegpunkte beschränkt sich auf vier Richtungen: jeweils zwei horizontal und vertikal.

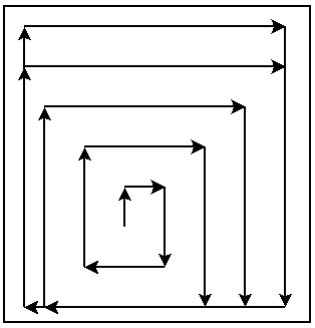
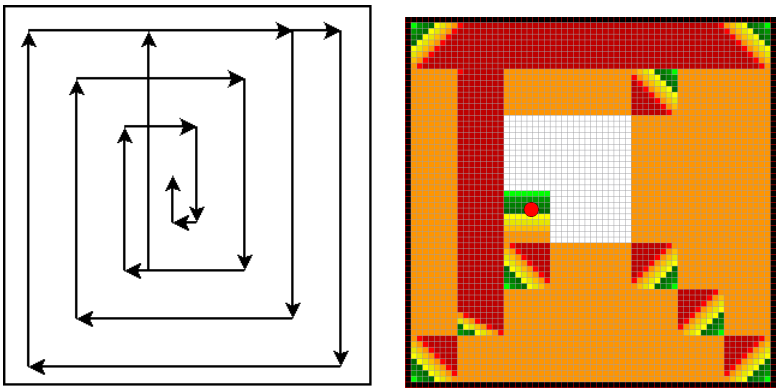


Abbildung 4.6.: Wegpunkte einer Breitensuche

Tiefensuche:



(a) (b)

Abbildung 4.7.: (a) Wegpunkte der Tiefensuche; (b) Doppelt betrachtete Wegpunkte

Für die Tiefensuche wurde das erste Schema der Tiefensuche aus Kapitel 3.3.2 noch leicht verändert. Der erste Wegpunkt ist vom Startpunkt direkt nach oben. Dann wird im Uhrzeigersinn jeweils so weit geradeaus geflogen, bis ein Hindernis auftaucht. Dabei wird die Größe der Karte in

Schrittgröße gespeichert. Die Schritte werden für die horziontale und vertikale Richtung seperat gespeichert und sind von der eingestellten Sichtweite abhängig. In der weiteren Generierung wird diese Schrittgröße jeweils um eins verringert. Dies führt dazu, dass immer kleinere Kreise gezogen werden, bis die Mitte der Karte erreicht ist. Es werden lediglich die Zellen vom Startpunkt nach oben und von dort nach rechts doppelt betrachtet (Abbildung 4.7).

Chaosdrive:

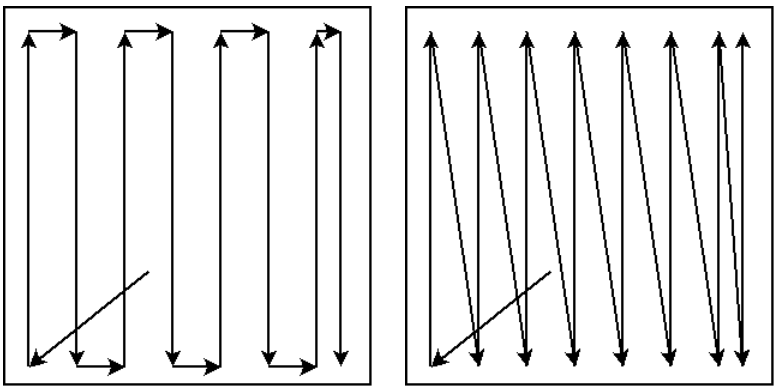
Der Chaosdrive-Algorithmus geht im Grunde genommen wie in Kapitel 2.2.3 beschrieben vor. Der Winkel, um sich von der Wand weg zu bewegen, wird aus einem Bereich von 90 ° bis 270 ° zusätzlich zum aktuellen Winkel zufällig berechnet. Die Wegpunkte werden beim Erreichen der Wand gespeichert und sind vollkommen von den Parametern Sichtbereich und Toleranz unabhängig. Der Sichtbereich beeinflusst lediglich das Einzeichnen in der Grafik. Der Algorithmus hat keine Abbruchbedingung, und wird so lange ausgeführt, bis der Benutzer den Pause- oder Reset-Button drückt. Sobald dies geschieht werden die generierten Wegpunkte gespeichert und können an den Quadrokopter geschickt werden. Beim Pausieren wird die aktuelle Position ebenfalls als Wegpunkt gespeichert. Beim Fortfahren wird wieder eine zufällige neue Richtung ausgewählt.

MyChaosdrive:

Angelehnt an der Chaosdrive-Idee, wurde dieser Algorithmus anhand der Überlegungen mit der Beschränkung auf acht Richtungen entworfen. Man hat als zufällige Parameter einerseits eine der acht Richtungen (horizontal,vertikal,diagonal) und andererseits eine Variable, die die Länge der nächsten Bewegung vorgibt. Diese zweite Variable wurde deshalb gewählt, da der Algorithmus bei nur 8 zufälligen Richtungen lediglich an den vier Wänden und in den Raumdiagonalen Wegpunkte generieren würde. Wie bei Chaosdrive gibt es für MyChaosdrive bis auf die Benutzereingabe keine Abbruchbedingung.

Zick-Zack

Das Zeilenweise abfliegen eines Raumes würde jeweils zwei Wegpunkte direkt nebeneinander erzeugen. Dies ist für den Quadrokopter nicht einfach. Deshalb liegt als Überlegung für ein triviales Abfliegen eines leeren Raumes nahe, ein Zick-Zack-Muster zu erzeugen. Dieses Muster



(a) Zeilenweise (b) Zick-Zack

Abbildung 4.8.: Vergleich (a) Zeilenweise- zu (b) Zick-Zack-Muster

bekommt als Startpunkt (beziehungsweise als ersten Wegpunkt) die linke untere Ecke des Raumes, geht von dort aus senkrecht nach oben und geht danach schräg nach unten bis zur nächsten Wand im Abstand von genau einem Sichtbereich zum links liegenden Wegpunkt. Dies wird so lange wiederholt, bis der ganze Bereich abgedeckt ist. (Abbildung 4.8)

4.3.2. Weitere iterative Suchalgorithmen

Die nachfolgenden Algorithmen wurden implementiert, aber ab einer gewissen Programmversion nicht weiter angepasst. Das heißt die Algorithmen funktionieren in der Abgabeversion des Programms nicht. Es gibt mehrere Gründe, warum die Algorithmen nicht weiter bearbeitet wurden:

- Die Suchalgorithmen haben bei weitem zu viele Zwischenschritte generiert. Die Schritte wurden iterativ generiert. Zur Übertragung auf den Quadrokopter ist eine Obergrenze an Wegpunkten eingestellt. Diese ist zwar veränderbar, aber bei einer Größenordnung von 500-1000 Wegpunkten ist es sinnlos, diese permanent anzupassen.

- Anhand der vorgegebenen Karte wurden die Suchalgorithmen selbst entworfen. Daher kann für diese keine Vollständigkeit gewährleistet werden. Die Laufzeit bis der Raum komplett abgesucht worden ist, kann bei manchen Algorithmen extrem lange dauern, da teilweise Wege “verbaut“ werden, indem man auf bereits besuchte Zellen nicht zurück geht.

- Die generierten Wegpunkte der Suchalgorithmen sind jeweils nur einen Zellenabstand voneinander entfernt. Diese Distanz ist bei Weitem zu kurz um die Wegpunkte sinnvoll mit

dem Quadrokopter abzufliegen. Zum Bestätigen, dass ein Wegpunkt erreicht wurde, hat der Quadrokopter einen eingestellten Wert. Dieser Wert war zum Zeitpunkt der Praxis-Versuche auf 20cm eingestellt. Die Zellgröße bei der Praxis war auf 3cm eingestellt. Das heißt beim geraudeaus Fliegen würden im Rahmen der 20cm Toleranz 6 Wegpunkte einfach übersprungen werden. Das geradeaus Fliegen ist in den hier betrachteten Suchalgorithmen jedoch eine Seltenheit, was zur Folge hat, dass die Wegpunkte wild verteilt angeflogen werden.

- Anfangs wurde bei der Generierung der Wegpunkte nicht darauf geachtet, dass der Quadrokopter ein so großes Überschwingen beim Abfliegen der Wegpunkte hat. Der nachfolgende Wegpunkt kann prompt direkt in der entgegengesetzten Richtung angegeben werden. Dies kann zum Beispiel bei einer Sackgasse passieren.

Für die Algorithmen MyBFS und MyDFS wurde die, in Kapitel 4.2 erklärte, Methode genutzt, die benachbarten Zellen zu betrachten und zu kontrollieren, wo der beste nächste Wegpunkt ist.

MyBFS:

Die Idee für MyBFS kam von der klassischen Breitensuche und hat sich mit der Grundlage der Karte entwickelt. Der Startpunkt der Suche wird gespeichert und zum Berechnen der aktuelle Distanz von diesem genutzt. Der Algorithmus betrachtet in jedem Schritt die benachbarten Zellen. Von den aufsummierten Werten wird eine bestimmte Anzahl der besten (diejenigen, welche den geringsten Wert haben) herausgepickt. Mithilfe der jetzt bekannten Richtungen wird der Abstand des dann auszuführenden Schrittes zum Startpunkt berechnet. Von diesen wird derjenige ausgewählt, der den geringsten Abstand zum Startpunkt hat. Das heißt man bewegt sich grundsätzlich immer von den Zellen weg, welche man bereits besucht hat, aber versucht gleichzeitig den Abstand zum Startpunkt klein zu halten. Es ergibt sich keine spiralförmige Bewegung in einem Drehsinn, sondern je nach Zellbelegung eine immer drehende Spiralbewegung.

MyDFS:

MyDFS ist im Grunde genommen nur eine kleine Veränderung zur MyBFS-Suche. Man versucht weiterhin von den bisher besuchten Zellen weg zu kommen, nur nutzt man dieses Mal die Distanz

zum Startpunkt um eine größere Entfernung zu diesem zu bekommen. Die Bewegung führt also dazu, dass man primär in eine Ecke des Raumes fährt. Diese wird dann spiralförmig ausgefüllt, so lange bis man an einer Kante entlang in die nächste Ecke gelangt. Sind alle Ecken ausgefüllt, fährt der Algorithmus spiralförmig in Richtung des Startpunktes. In ungünstigen Fällen versucht der Algorithmus über bereits besuchte Zellen wieder vom Startpunkt weg zu fahren.

Randomsearch:

Randomsearch wurde anfänglich genutzt um eine Bewegung in der Simulation zu erzeugen. Der Algorithmus nimmt zwei zufällig generierte Zahlen und rechnet diese auf einen der drei Werte -1,0,1 herunter. Dieser Wert wird mit der Zellgröße multipliziert und dann jeweils für eine X- und Y-Koordinate eingesetzt. Der Quadrokopter bewegt sich also immer um maximal eine Zelle in eine zufällige Richtung.

4.4. Kartenerstellungs-Tool

Das hier angefertigte Tool wird dazu verwendet um selbst Karten zu zeichnen. (Abbildung 4.9) Es wurde dafür entwickelt, um möglichst simpel und schnell eine Umgebung für einen beliebig komplexen Raum zu erstellen. Anhand der Karte können verschiedene Suchalgorithmen in der Simulation getestet werden, und auch die Wegpunkte für eine reale Karte generiert werden.

1:

Combobox, die für die Aktivität in der Grafik zuständig ist. Die Option “Startpunkt verschieben“ erlaubt den Quadrokopter (roter Punkt) per Drag-And-Drop auf eine beliebige Startposition zu verschieben. Dabei wird die Kollision des Objekts mit den Wänden (belegten Zellen) beachtet, so dass der Startpunkt nicht in einer Wand liegen kann.
Die selbe Möglichkeit gibt die Option “Suchobjekt verschieben“, jedoch wurde das Suchobjekt per Define aus der Grafik entfernt. Anfangs war es so gedacht, dass man die Position des zu findenden Objekts bereits voreinstellen kann. Jedoch wird im Allgemeinen davon ausgegangen, dass Suchobjekte zufällig im Raum verteilt sein können. Für die uninformierte Suche ist es obsolet, kann jedoch in Zukunft für heuristische Suchen wieder eingesetzt werden.
Des Weiteren ist als Option “Wände/Objekte erstellen“ auswählbar. Diese Option erlaubt Wän-

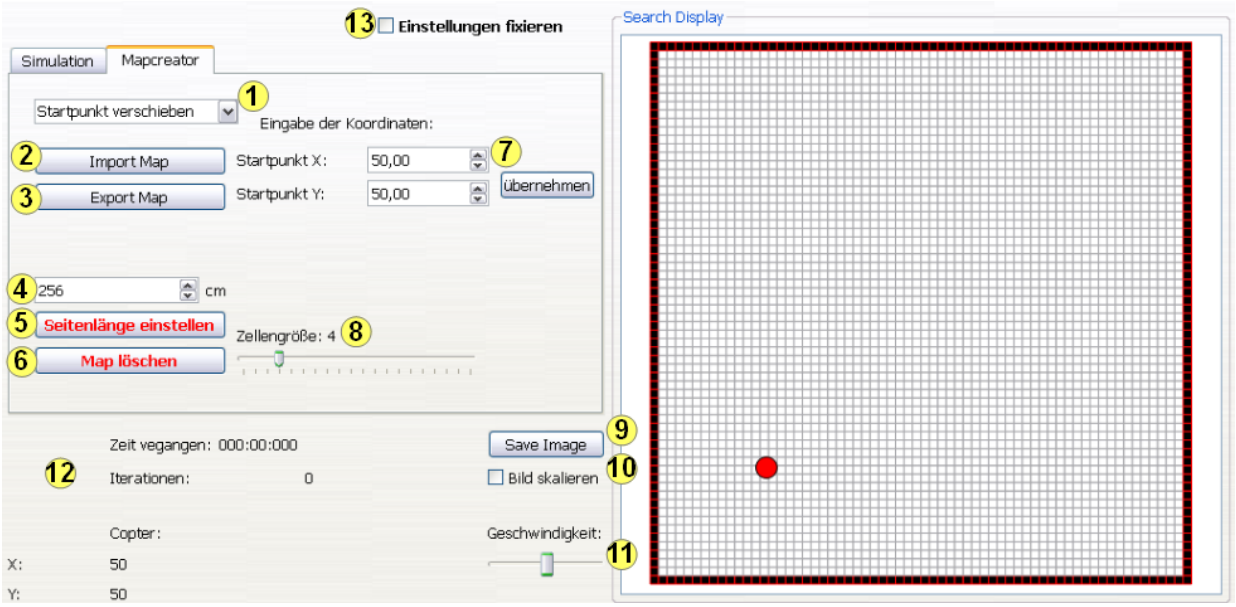


Abbildung 4.9.: MapCreator-Tab

de per Mausklick auf folgende Weise einzuzeichnen. Per linker Maustaste kann man Zellen als belegt oder frei markieren. Eine Zelle wird als belegt markiert, wenn sie vorher frei war, und umgekehrt. Wenn man die Taste gedrückt lässt und die Maus bewegt, werden die jeweiligen Zellen im selben Zustand wie die zuerst angeklickte Zelle markiert. Das heißt wenn zum Beispiel die erste Zelle per Linksklick als frei markiert wird und man bewegt die Maus bei gedrückter linker Maustaste weiter, werden alle anderen berührten Zellen auch als frei markiert.

2:

Ermöglicht eine bereits gespeicherte oder erzeugte Karte zu importieren. Aktualisiert die Grafik.

3:

Speichert die aktuell sichtbare Karte in einer .map Datei [Schmitt, 2012].

4:

Wird als Einstellung für die Seitenlänge der Karte verwendet. Die Seitenlänge ist dabei abhängig von der minimalen Größe an Teil-Karten, welche sich auf 64 Zellen beläuft. Als maximaler Wert ist das dreifache der minimalen Seitenlänge (64∗Zellgröße) eingestellt, da zum Zeitpunkt der Implementierung die Zeichenmethode der Grafik jede Zelle als einzelnes Grafikobjekt verwaltet hat. Bereits bei dieser maximalen Größe gab es beim Erstellen der Grafik deutliche Verzögerungen.

5:

Löscht die aktuelle Karte und übernimmt die Einstellung von **4**.

6:

Löscht die aktuelle Karte und erzeugt eine neue Standard-Karte. Die Standard-Karte hat die äußersten Zellen belegt, eine Zellgröße von 4cm und eine Seitenlänge von 256cm in einer quadratischen Form.

7:

Neben der Variante den Startpunkt per Drag-And-Drop zu setzen (siehe **1**), kann man die Startposition direkt eingeben und einstellen.

8:

Verändert die Zellgröße der aktuellen Karte. Dabei wird nicht die Karte gelöscht, sondern lediglich die Größe verändert. Alle als belegt markierten Zellen bleiben auch markiert.

9-12

Siehe Kapitel 4.3.

13

Verhindert, dass weitere Einstellungen an der Karte beziehungsweise der Startposition vorgenommen werden. Aktiviert im Simulationstab den Start-Button.

5. Evaluierung

5.1. Überblick

In den nachfolgenden Kapiteln werden drei ausgewählte Algorithmen sowohl in der Software als auch in der Praxis ausgewertet. Betrachtet werden Breitensuche, Tiefensuche und ein Zick-Zack-Muster. Im Kapitel Software (5.2) wird darauf eingegangen, wie die Wegpunkte am Computer für den Quadrokopter erstellt werden. Die praktischen Versuche (Kapitel 5.3) zeigen in Form von grafisch aufgetragenen X- und Y-Werten die Flugbahn des Quadrokopters.

5.2. Simulation

Die Evaluierung der Simulation beschränkt sich darauf zu testen, ob die Wegpunkte am Computer richtig generiert werden, damit diese dem Quadrokopter übergeben werden können und dort weiter benutzt werden.

Das Ändern der Zellgröße, verändert die gesamte Kartengröße. Dies kommt daher, dass die Karte [Schmitt, 2012] eine minimal Anzahl von Zellen hat (hier: 64). Das heißt die kleinste Größe der Karte ist 64∗Zellgröße, wobei die Zellgröße in einem Wertebereich von 1...20cm ist [Schmitt, 2012].

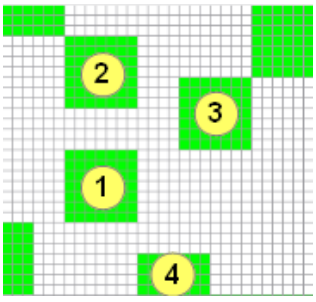


Abbildung 5.1.: Wegpunkte mit Toleranzverschiebung bei einer Breitensuche

Weiterhin lässt sich der Sichtbereich des Quadrokopters variieren. Daraus folgt eine Veränderung der Wegpunktgenerierung. Je größer der Sichtbereich eingestellt ist, desto weiter sind die Wegpunkte voneinander entfernt, et vice versa.

Ein weiterer Parameter ist die Toleranz der Wegpunkte. Dieser Parameter verschiebt die Wegpunkte in Flugrichtung um den entsprechenden Wert weiter. Der je-

weils nachfolgende Wegpunkt wird dabei jedoch nicht in die selbe Richtung verschoben! Die Auswirkung dieser Verschiebung wird für jeden Algorithmus einzeln betrachtet. Abbildung 5.1 zeigt ein Beispiel für die Breitensuche mit Toleranzverschiebung. Hintergrund dieser Verschiebung ist, dass der Quadrokopter beim Flug ungefähr 20cm vor dem Erreichen des Wegpunkt ebendiesen bereits als erreicht markiert. Man kann ohne den Code des Quadrokopters ändern zu müssen, mit dem Toleranz-Parameter testen, wie genau der Quadrokopter die Wegpunkte in der Realität anfliegt. Dies ist wichtig um den abzusuchenden Bereich vollständig abzudecken.

Die Wegpunktgenerierung ist ebenfalls abhängig von der Lage des Startpunktes. Dies hat für jeden Algorithmus individuell Auswirkungen und wird im jeweils entsprechenden Kapitel betrachtet. Dadurch, dass der Startpunkt rasterlos verschiebbar ist, werden teilweise Zellen, die nur gerade so berührt werden, bereits als betrachtet markiert. Dies kann in ungünstigen Fällen dazu führen, dass Objekte zwischen zwei Wegpunktstrecken nicht erkannt werden, da sie von beiden Seiten als betrachtet markiert worden sind, jedoch nicht wirklich komplett gescannt worden sind.

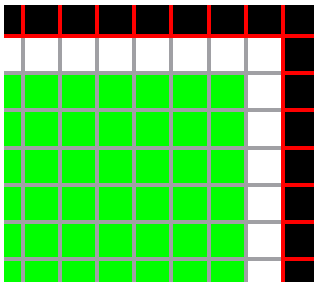


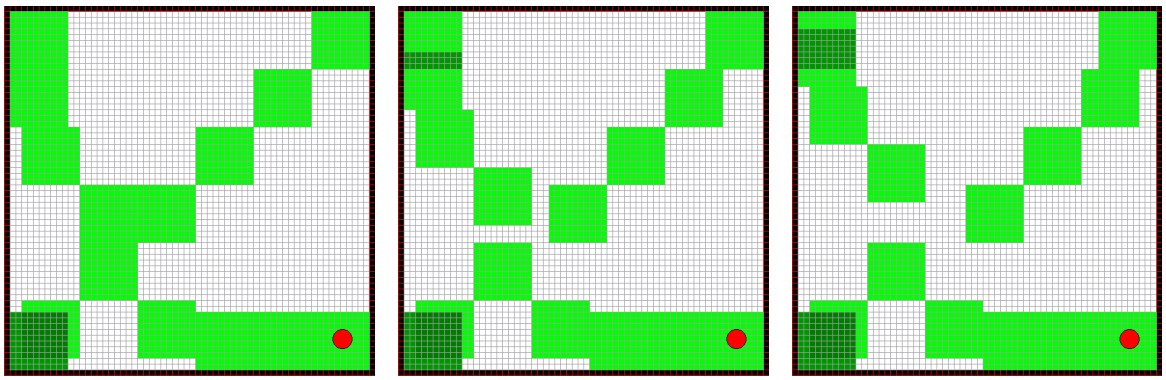
Abbildung 5.2.: Zellen am oberen und rechten Rand nicht betrachtet

Beim Testen hat sich gezeigt, dass die Wegpunkte teilweise am oberen und rechten Rand der Karte eine Zelle zu wenig betrachten. (siehe Abbildung 5.2) Dies liegt zu großer Wahrscheinlichkeit an der internen Kollisionserkennung des Grafikobjekts für den Quadrokopter. Diese erlaubt nicht den Quadrokopter auf eine Wand zu bewegen und fragt unter Umständen eine Zellenreihe zu viel ab.

5.2.1. Breitensuche

Die Toleranzverschiebung wirkt sich bei der Breitensuche folgendermaßen aus: Da die Wegpunkte spiralförmig vom Startpunkt wegführen wird jeder einzelne Wegpunkt in Flugrichtung um den Toleranzbereich geraudeaus “geschoben“. Diese Verschiebung wirkt sich jedoch nur für diesen einzelnen Wegpunkt aus und verändert nicht die Position der anderen Wegpunkte. Der Vergleich der Abbildungen 5.3 zeigt die Auswirkung der Verschiebung.

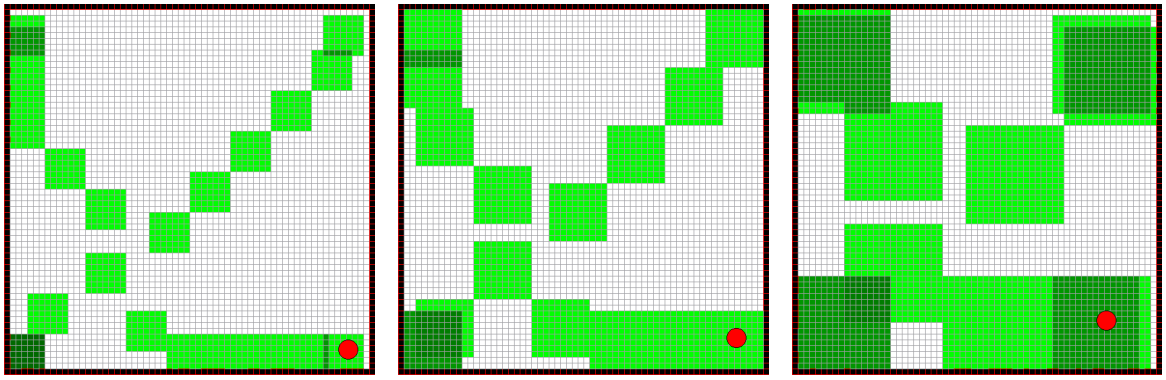
Die Veränderung des Sichtbereichs wurde in der Praxis mit einem für die Toleranz fixen Wert von 10cm getestet, deshalb sieht man in Abbildung 5.4 den Vergleich der in der Software erstellten Wegpunkte.



(a) (b) (c)

Abbildung 5.3.: Breitensuche mit Sichtbereich = 30cm, Toleranz = (a) 0cm, (b) 10cm, (c) 20cm

Die Lage des Startpunktes hat für die Wegpunkte der Breitensuche keine besonderen Auswirkungen. Nach dem hier implementierten Algorithmus ist es lediglich ungünstig, dass die Wegpunkte über bereits betrachteten Wegen nochmals liegen. Bei einem länglichen, rechteckigen Raum beispielsweise würden die Strecken am Rand extrem oft mehrfach abgedeckt werden.



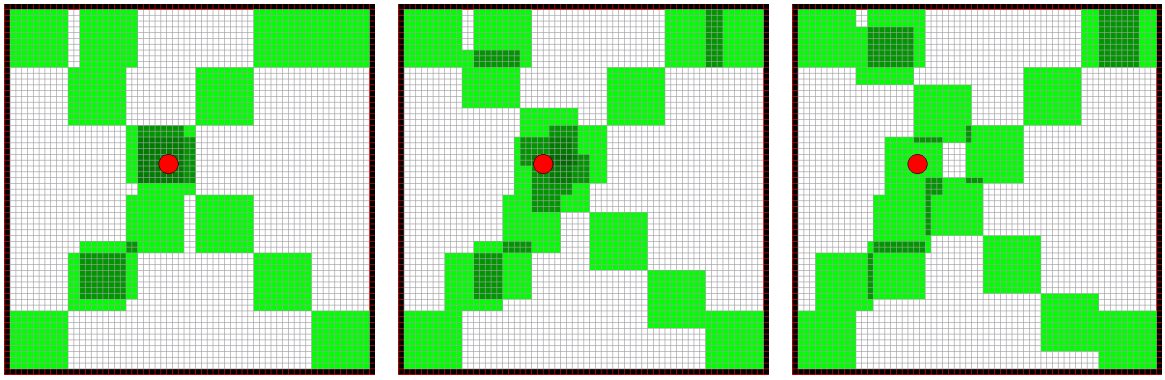
(a) (b) (c)

Abbildung 5.4.: Breitensuche mit Sichtbereich = (a) 21cm, (b) 30cm, (c) 51cm, Toleranz = 10cm

5.2.2. Tiefensuche

Ebenso wie bei der Breitensuche werden die Wegpunkte mit der Toleranz weiter in Flugrichtung nach vorne versetzt. Die Wegpunkte werden wie in Abbildung 5.5 generiert und wirken verwirrend, da die Startposition und der davon ausgehende nächste Wegpunkt senkrecht nach oben eingezeichnet sind. Der Startpunkt befindet sich wie bei der Breitensuche links unten (Koordinaten 50/50 bei Gesamtgröße von 192/192). Bei Abbildung 5.5 (c) werden die Wegpunkte direkt in der Mitte der Karte nicht exakt erstellt. Die Toleranzverschiebung wirkt sich nicht auf

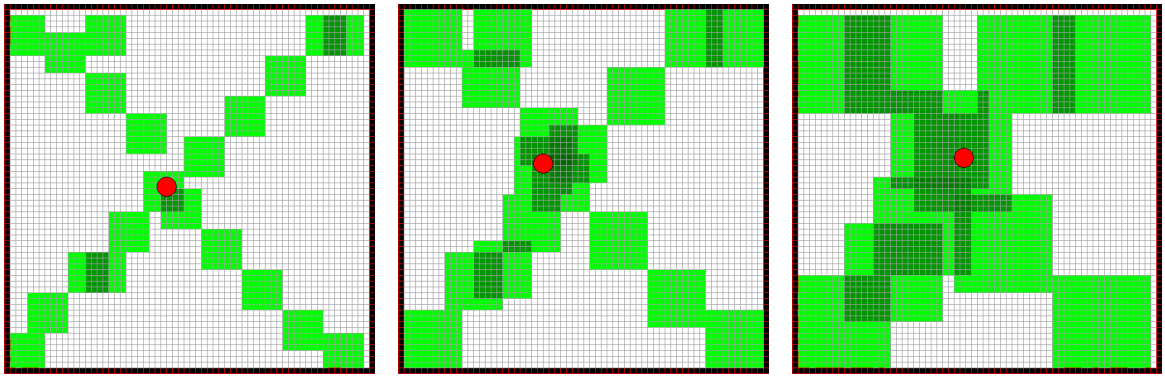
die Abbruchbedingung der Wegpunkterstellung aus. Das heißt der letzte Wegpunkt wird als “im Zentrum“ erkannt, jedoch noch um die Toleranz weiter verschoben.



(a) (b) (c)

Abbildung 5.5.: Tiefensuche mit Sichtbereich = 30cm, Toleranz = (a) 0cm, (b) 10cm, (c) 20cm

Für den Sichtbereich wurden ebenfalls die selben Werte wie bei der Breitensuche verwendet (Abbildung 5.6).



(a) (b) (c)

Abbildung 5.6.: Tiefensuche mit Sichtbereich = (a) 21cm, (b) 30cm, (c) 51cm, Toleranz = 10cm

Der Startpunkt hat für die Tiefensuche deutlich weniger Auswirkungen als für die Breitensuche, da die Wegpunkte der Tiefensuche immer gleich sind. Es wird vom Startpunkt aus senkrecht nach oben und von dort nach rechts jeweils ein Wegpunkt generiert und ab diesem Wegpunkt würde für jeden beliebigen Startpunkt die selbe Flugbahn verfolgt werden.

5.2.3. Zick-Zack-Muster

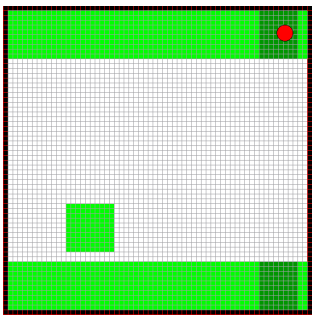


Abbildung 5.7.: Zick-Zack mit Sichtbereich=30cm, Toleranz=0cm

Bei einem simplen Muster wie diesem Zick-Zack eine Toleranzverschiebung einzustellen, macht nur bedingt Sinn. Im hier implementierten Code würden die Wegpunkte außerhalb des Bereichs der Karte, also in eine Wand verschoben werden. Wegen der Kollisionsabfrage des Grafikobjekts erlaubt die Simulation nicht, die Wegpunkte einzuzeichnen. Aus diesem Grund kann hier keine grafische Auswertung für dieses Muster erfolgen. Die Wegpunkte werden trotzdem erstellt, da der Toleranzparameter lediglich zum Testen für das Abfliegen der Wegpunkte dient. Im Regelfall wird davon abgeraten beim Zick-Zack-Muster eine Toleranz größer als 0 einzustellen. Der erste Wegpunkt (links unten) bleibt unabhängig vom Startpunkt immer der gleiche und wird mit der Toleranz nicht nur nach unten sondern auch nach links verschoben, um das Erreichen des Quadrokopters zu testen. Durch die Veränderung des Sichtbereichs werden mehr (kleiner Sichtbereich) beziehungsweise weniger (großer Sichtbereich) Wegpunkte erstellt. Abbildung 5.7 zeigt beispielhaft die Wegpunkte des Zick-Zack-Musters. Daran zu sehen ist, dass die rechten Wegpunkte sich teilweise überschneiden, da der äußere Rand sonst nicht komplett betrachtet werden würde.

5.3. Praxis

Für die Auswertung der Flugbahn hat man drei verschiedene Anhaltspunkte: Die Wegpunkte selbst, die Odometrie und das optische Tracking-System. Die Wegpunkte wurden von den Telemetriedaten des Quadrokopters als nächster Soll-Wegpunkt empfangen. Die Odometrie ist die Position des Quadrokopters, welche er selbst anhand des optischen Flusssensors berechnet hat. Das optische Tracking arbeitet mit vier Infrarot-Kameras, welche den Quadrokopter mithilfe eines Infrarot-Senders erkennen. Zur Auswertung ist der wichtigste Vergleich zwischen den Koordinaten der Wegpunkte mit den Daten des optischen Trackings, da die Odometrie lediglich die internen Koordinaten des Quadrokopters verwaltet.

Für die Versuchsanordnung wurden Wegpunkte mit verschiedenen Parametern erzeugt. Die Einstellung der Karte wurde fest am Anfang eingestellt. Die Zellgröße wurde auf 3cm fixiert. Daraus ergibt sich eine Kartengröße von 1,92m$*$1,92m. Der Sichtbereich berechnet sich aus der Kartengröße abzüglich der jeweils äußeren Zellen, welche im GUI als Wand markiert sind. Die

Startposition des Quadrokopters liegt bei 50cm/50cm. Das interne Koordinatensystem hat den Ursprung von X und Y in der linken unteren Ecke. Y läuft in positiver Richtung nach oben, X nach rechts. Die generierten Wegpunkte belaufen sich *ungefähr* in einem X- und Y-Bereich von 3cm bis 189cm. *Ungefähr* deshalb, weil der Sichtbereich und die Toleranz je nach Algorithmus teilweise Werte außerhalb beziehungsweise Werte weiter innerhalb dieses Bereichs erzeugen.
Da parallel zum Abfliegen der Wegpunkte keine Kollisionserkennung auf dem Quadrokopter gelaufen ist, wurden die Versuche ohne Wände in einem größeren Bereich abgeflogen. Im Allgemeinen ist der Quadrokopter auch oft aus dem Bereich der Karte heraus geflogen. Dies hat zur Folge, dass man in späteren Versuchreihen mit Wänden entweder die Wegpunkte in größerem Abstand zur Wand erstellen, oder die Wegpunkte mit Rückkopplung von der Kollsionserkennung nachbessern müsste.
Wie in Kapitel 5.2 wurden bei den Praxis-Versuchen die Parameter Sichtweite und Toleranz verändert. Es wurden insgesamt 23 Messungen durchgeführt. Dabei wurden teilweise die selben Parametereinstellung verwendet. Als Vergleichsmessungen sind jeweils mit einer Toleranz von 10cm die Sichtweiten von 21cm, 30cm und 51cm getestet worden. Weiterhin wurde die Toleranz von 0cm und 20cm bei einer festen Sichtweite von 30cm für jeden Algorithmus getestet. Im Folgenden werden nur einzelne Beispiele gezeigt, alle Messungen befinden sich im Anhang.

5.3.1. Breitensuche

In Abbildung 5.8 ist der Quadrokopter sehr deutlich aus dem abgesteckten Bereich heraus geflogen. Man kann erkennen, dass er ungefähr die Wegpunkte angeflogen ist, jedoch sind teilweise starke Schwankungen vorhanden, die dann eine große nicht betrachtete Lücke zurück lassen. Dies ist besonders deutlich beim letzten Wegpunkt (rechts unten) zu erkennen. Dort ist der Sichtbereich von 21cm zwischen den zwei Wegpunktpfaden definitv nicht abgedeckt. Die Lücke dort ist ungefähr 50cm groß, das beträgt mehr als das doppelte vom Soll-Wert! Der überschwingende Bogen des Quadrokopters kommt daher, dass der Quadrokopter eine längere gerade Strecke zu Fliegen hatte als vorher und deshalb noch eine Restbeschleunigung übrig bleibt. Eine ähnliche Kurve sieht man auch auf der linken Seite.

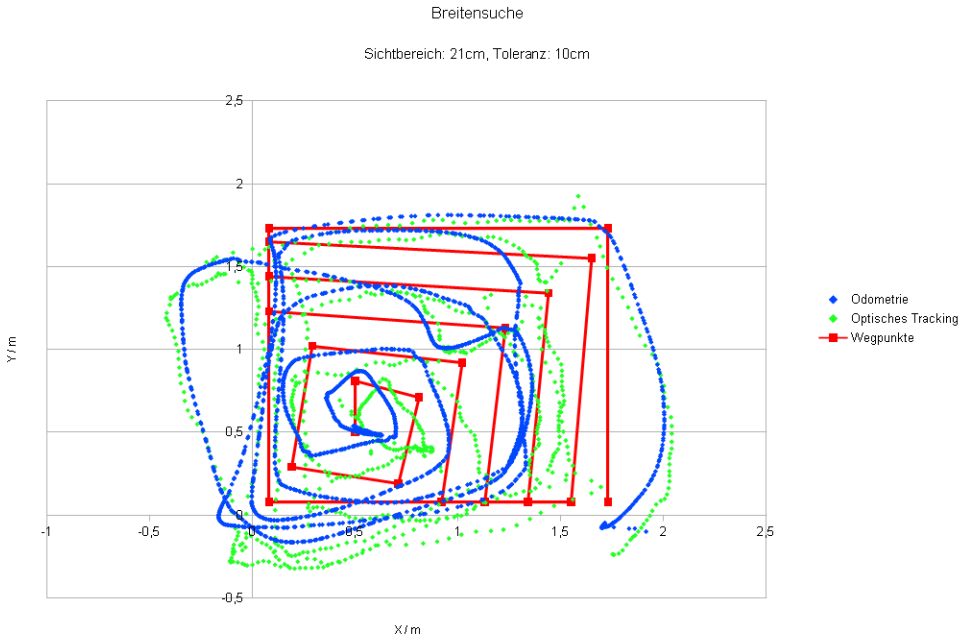


Abbildung 5.8.: Breitensuche mit Sichtbereich = 21cm, Toleranz = 10cm

Abbildung 5.9 zeigt den Flug bei einem größeren Sichtbereich. Offensichtlich liegt hier ein deutlicher Fehler in der Odometrie des Quadrokopters vor. Beim Vergleich der Daten des Optischen Trackings mit den Wegpunkten ist jedoch ein sehr schönes Ergebnis vorzuzeigen. Der Quadrokopter ist weiterhin deutlich über die Wegpunkte hinausgeflogen, die tatsächliche Position ist allerdings sehr nahe am gewünschten Wegpunkt. Der abgedeckte Sichtbereich ist überall ziemlich im Rahmen des Sollwertes, nur im rechten Teil ist eine größere Lücke.

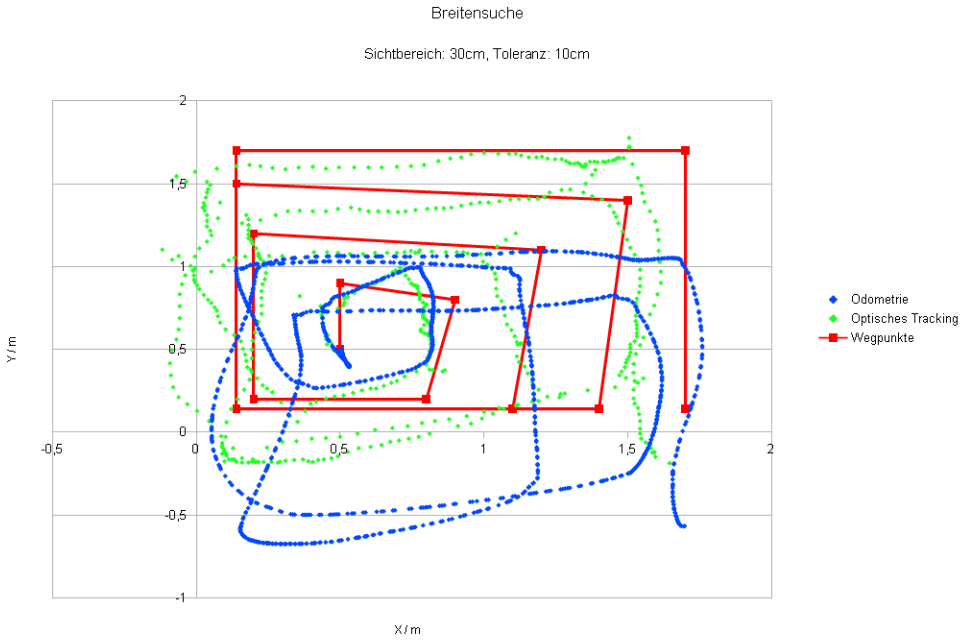


Abbildung 5.9.: Breitensuche mit Sichtbereich = 30cm, Toleranz = 10cm

Das Überschwingen des Quadrokopters durch die lange gerade Bewegung zuvor erkennt man besonders gut in Abbildung 5.10. Betrachtet man den Sichtbereich, merkt man, dass dieser sehr gut abgedeckt ist.

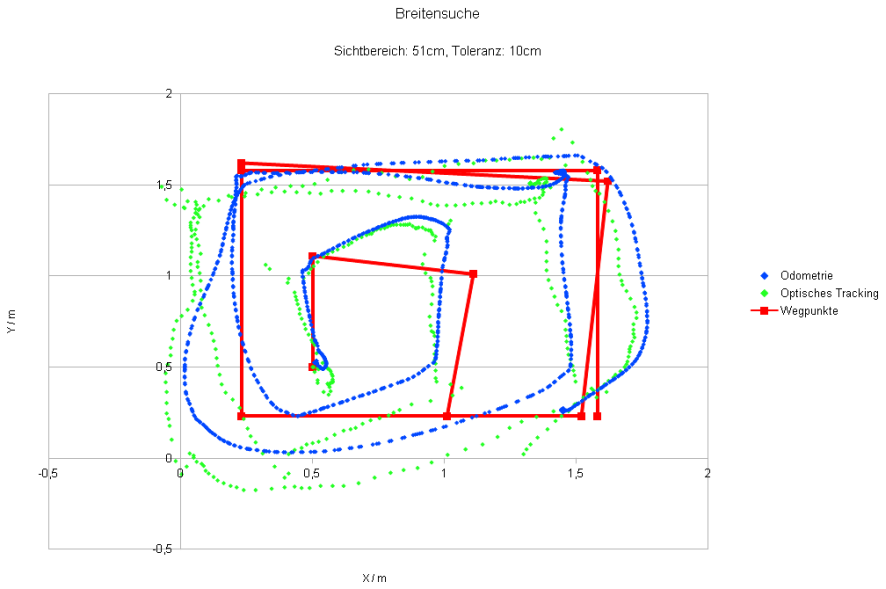


Abbildung 5.10.: Breitensuche mit Sichtbereich = 51cm, Toleranz = 10cm

Bei einem Toleranzbereich von 0cm liegen die Wegpunkte auf komplett geraden Strecken. (Abbildung 5.11) Offensichtlich kann der Quadrokopter auch diese Wegpunkte nicht im abgesteckten Rahmen ideal abfliegen. Laut der Odometrie-Kurve schien der Quadrokopter verhältnismäßig gut die Wegpunkte abzuarbeiten. Das optische Tracking jedoch zeigt, dass der Quadrokopter nach links deutlich über das Ziel hinaus geschossen ist. Der Sichtbereich sollte sich bis auf den äußersten Kreis im Rahmen von 30cm befinden.

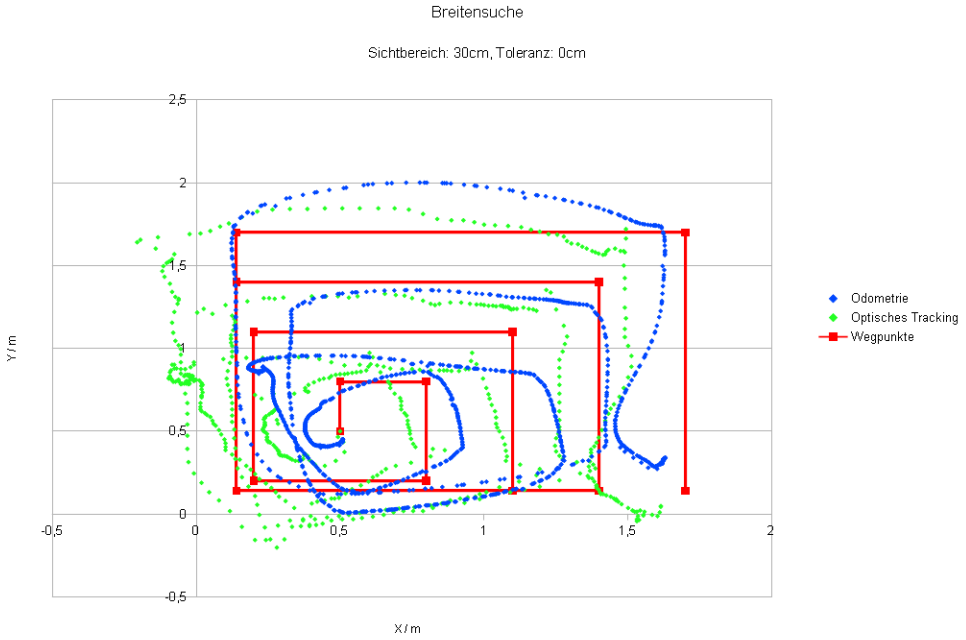


Abbildung 5.11.: Breitensuche mit Sichtbereich = 30cm, Toleranz = 0cm

Bei einer Toleranz von 20cm verhält sich der Quadrokopter relativ ähnlich wie zuvor bereits beschrieben. (Abbildung 5.12) Er bewegt sich außerhalb des Bereichs von 0cm bis 192cm und hat ein deutliches Überschwingen nach Erreichen der Wegpunkte. Der Sichtbereich ist relativ schlecht abgedeckt, wie man an den großen weißen Flächen links vom Startpunkt und etwas weiter rechts gut erkennen kann.

Zusammenfassend lässt sich sagen, dass der abzusuchende Bereich für die Erstellung der Wegpunkte deutlich kleiner gewählt werden sollte, als der tatsächlich abzufliegende Bereich, da der Quadrokopter während des Fluges deutlich über die Wegpunkte hinaus fliegt. Die Flugbahn mit dem großen Sichtbereich von 51cm scheint am konstantesten zu sein. Durch die vielen Änderungen der Flugbahn mit mehreren Wegpunkten kommen Fehler in die Odometrie. Je nach Kamerawinkel sollte der Sichtbereich vermutlich trotzdem kleiner als der tatsächliche Kamerawinkel

gewählt werden, damit man einen gewissen Spielraum hat. Dabei zu beachten ist, dass man Objekte so mehrfach finden kann. Die Toleranz hat die Flugbahn nicht wie gedacht zum Positiven hin verändert. Das Problem mit dem Überschwingen wird dadurch nicht behoben, sondern eher verstärkt.

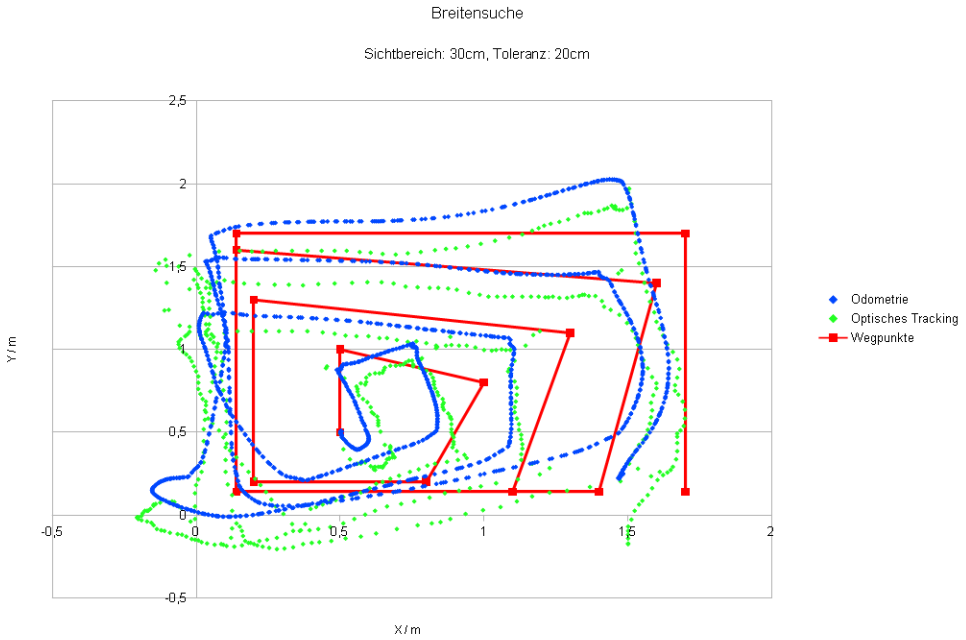


Abbildung 5.12.: Breitensuche mit Sichtbereich = 30cm, Toleranz = 20cm

5.3.2. Tiefensuche

Wie bereits für die Breitensuche in Kapitel 5.3.1 festgestellt, fliegt der Quadrokopter nach langen geraden Strecken über das Ziel hinaus. Dies sieht man bei allen nachfolgenden Abbildungen vor allem für die äußersten Kreise, die nach dem Startpunkt direkt abgeflogen werden.

In Abbildung 5.13 gibt es leichte Unterschiede zwischen den Werten der Odometrie und des optischen Trackings. Dies führt dazu dass ein sehr großer Bereich überhaupt nicht abgedeckt worden ist! Im zweiten Ring von außen hat sich der Quadrokopter laut optischem Tracking nicht befunden. Der Bereich in der Mitte der Karte hingegen wurde übermäßig gut abgedeckt.

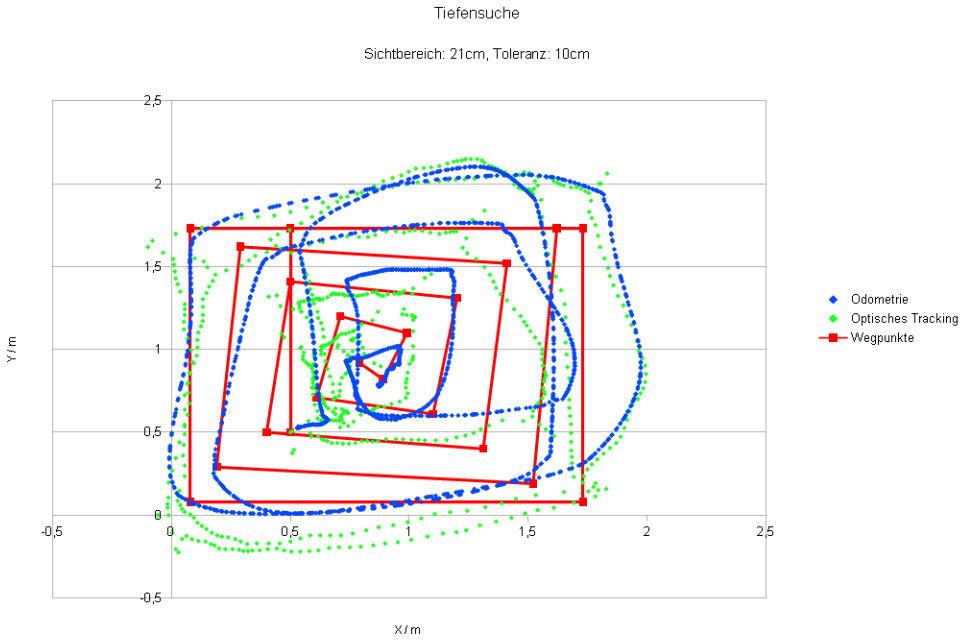


Abbildung 5.13.: Tiefensuche mit Sichtbereich = 21cm, Toleranz = 10cm

Der Quadrokopter hat in Abbildung 5.14 die Wegpunkte zwar nicht akurat angeflogen, jedoch in Betrachtung des Sichtbereichs bis auf den äußersten Ring alles gut abgedeckt.

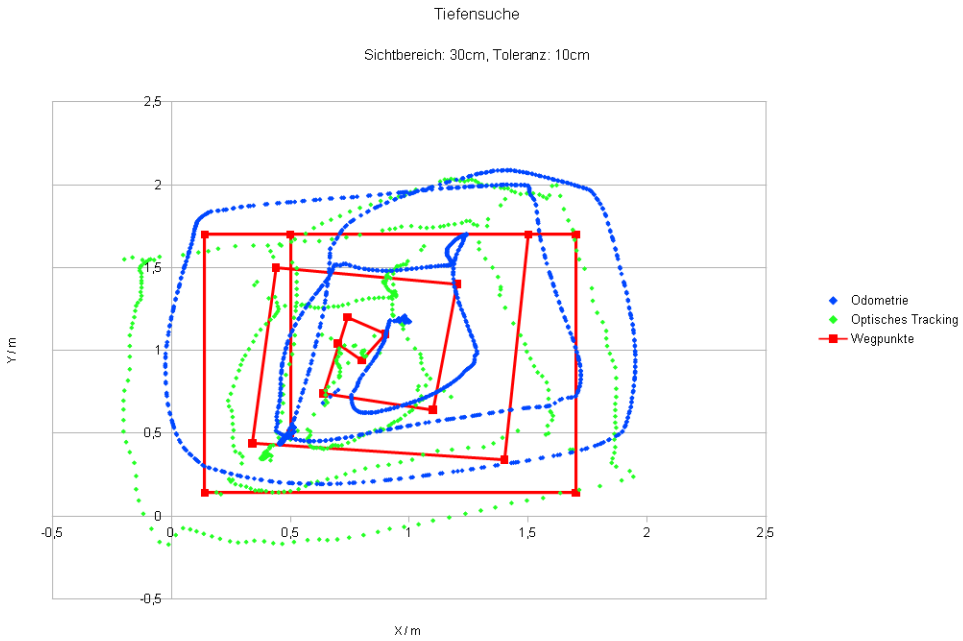


Abbildung 5.14.: Tiefensuche mit Sichtbereich = 30cm, Toleranz = 10cm

Der Versuch in Abbildung 5.15 zeigt, dass der Quadrokopter teilweise Wegpunkte zu früh "abhakt". Die Odometrie Kurve weicht deutlich von den Wegpunkten ab, wohingegen der Qua-

drokopter per optischen Tracking relativ gut die Wegpunkte verfolgt hat. Für den eingestellten Sichtbereich ist das Ergebnis gut.

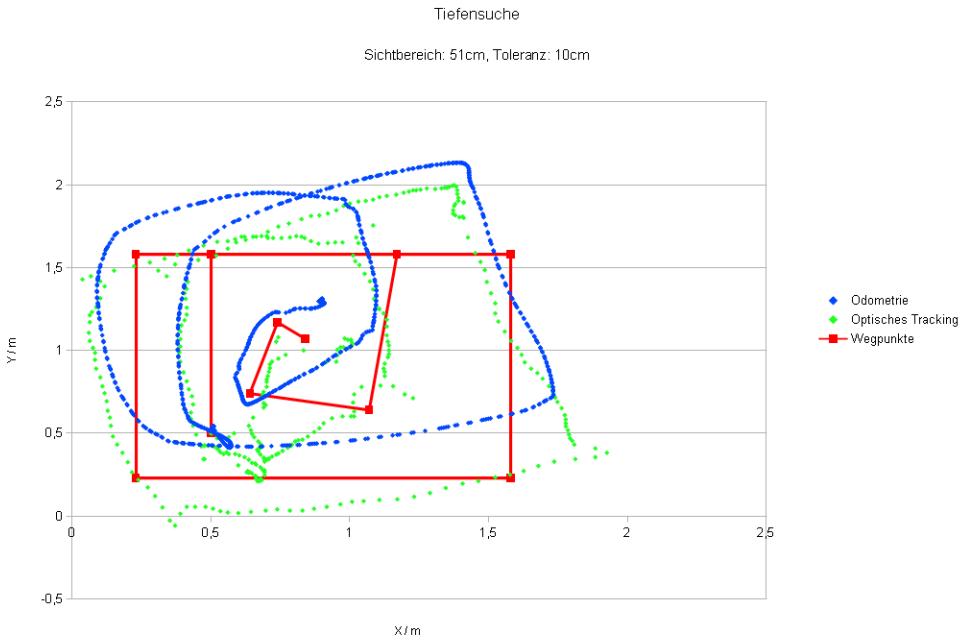


Abbildung 5.15.: Tiefensuche mit Sichtbereich = 51cm, Toleranz = 10cm

Dadurch, dass Odometrie und optisches Tracking durch Messfehler in der Odometrie teilweise unterschiedlich sind, wird in Abbildung 5.16 wieder ein größerer Bereich nicht richtig betrachtet. In diesem Bereich, rechts im Bild, sieht man die Odometrie genau in der Mitte, jedoch keinen Wert vom optischen Tracking.

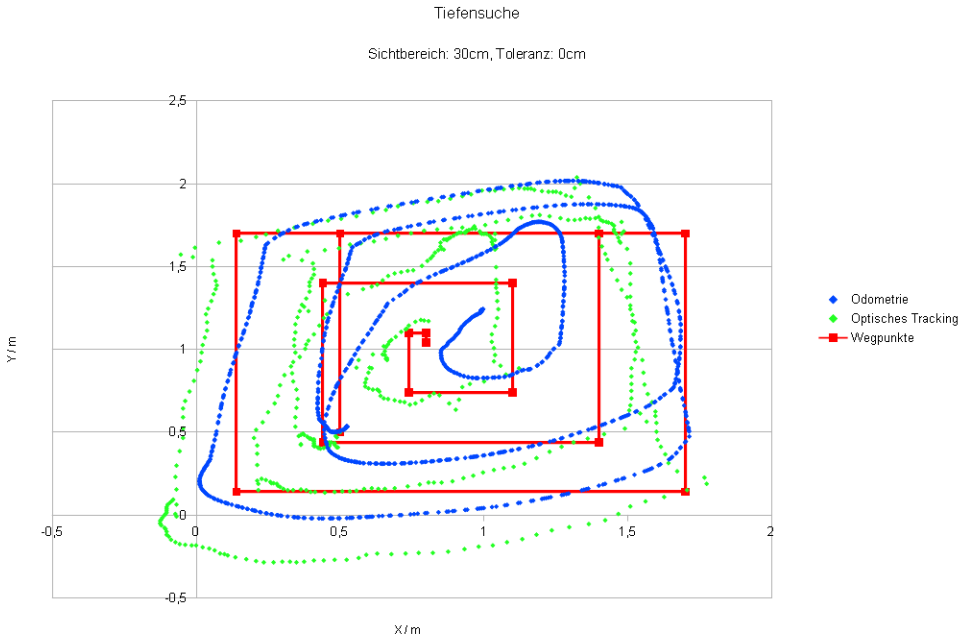


Abbildung 5.16.: Tiefensuche mit Sichtbereich = 30cm, Toleranz = 0cm

Im Versuch aus Abbildung 5.17 wird der Sichtbereich relativ gut abgedeckt. Hier bleibt lediglich das Problem, dass der Quadrokopter aus dem Bereich der Karte hinaus fliegt.
Zusammenfassend lässt sich ähnlich der Breitensuche sagen, dass die Wegpunkte für einen kleineren Bereich generiert werden sollten. Eventuell wäre es sinnvoll die äußersten Wegpunkte etwas zurückzusetzen, damit der Quadrokopter mehr Platz zum Überschwingen hat. Der Sichtbereich wurde bei einer Einstellung von 30cm am besten abgedeckt. Der Toleranzparameter hat wie bei der Breitensuche lediglich das Überschwingen verstärkt.

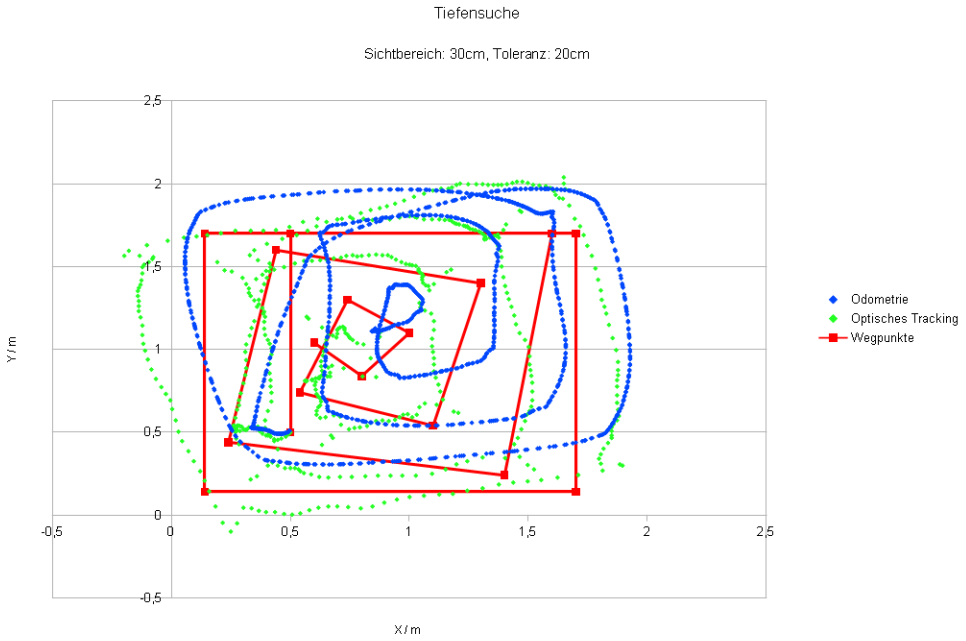


Abbildung 5.17.: Tiefensuche mit Sichtbereich = 30cm, Toleranz = 20cm

5.3.3. Zick-Zack-Muster

Das Zick-Zack-Muster hat von den drei getesteten Algorithmen mit Abstand am schlechtesten abgeschnitten. Die Anordnung der Wegpunkte war offensichtlich extrem ungünstig mit dem Anfliegen von Koordinaten des Quadrokopters umsetzbar.

In Abbildung 5.18 ist zwar der Suchbereich relativ gut abgedeckt, jedoch ist der Quadrokopter gleich am Anfang extrem weit aus diesem Bereich herausgeflogen. Am Ende ist der Quadrokopter ebenfalls deutlich zu weit außerhalb geflogen. Zusätzlich dazu ist der Quadrokopter den Bereich nicht zeilenweise abgeflogen, sondern hat immer wieder merkwürdige Kurvenbewegungen ausgeführt.

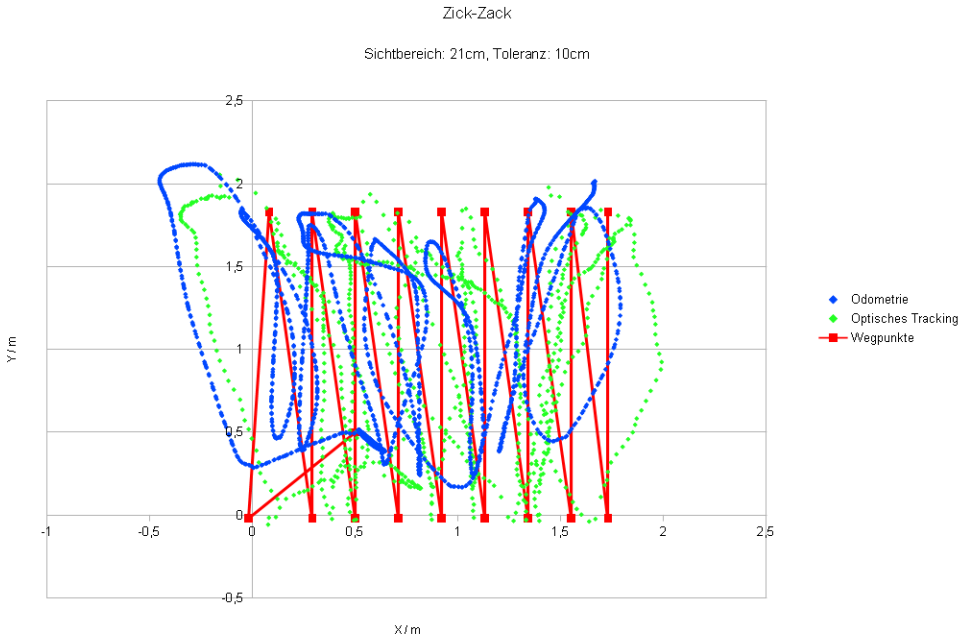


Abbildung 5.18.: Zick-Zack mit Sichtbereich = 21cm, Toleranz = 10cm

Neben dem bereits bekannten Überschwingen ist der Quadrokopter in Abbildung 5.19 eine extrem merkwürdige Flugbahn abgeflogen. Teilweise wurden Bereiche doppelt abgesucht, dafür wurden andere Bereiche überhaupt nicht betrachtet, zum Beispiel gleich am Anfang (oben links).

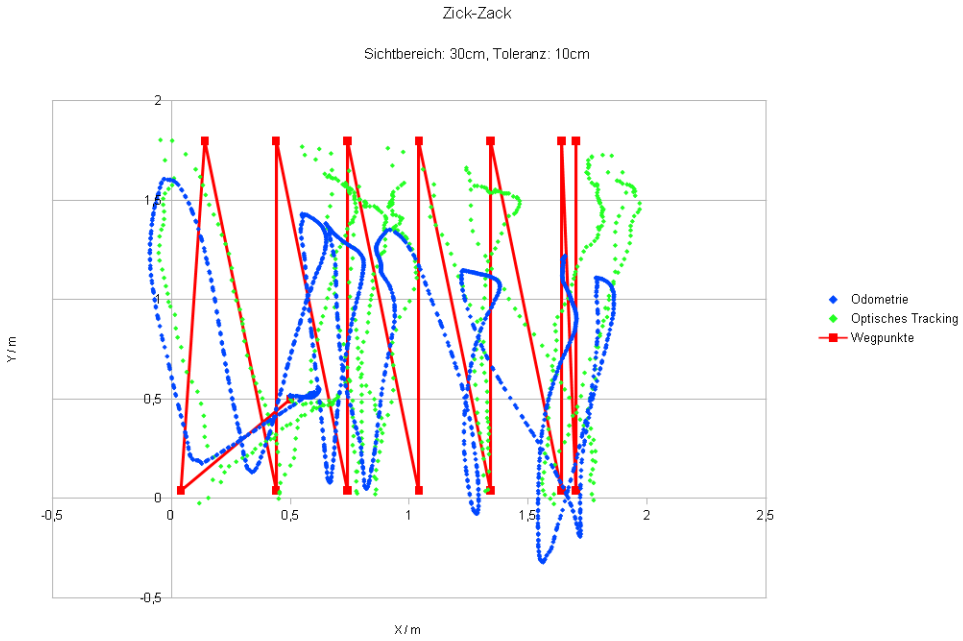


Abbildung 5.19.: Zick-Zack mit Sichtbereich = 30cm, Toleranz = 10cm

Bis auf ein kleines Überschwingen am Ende wird in Abbildung 5.20 relativ gut der Bereich abgesucht. Hier kann man gut erkennen, dass die Odometrie bereits ein ganzes Stück vor dem Erreichen des Wegpunktes umdreht. Dadurch, dass der letzte Wegpunkt weit außerhalb des Suchbereichs angeflogen worden ist, ist die Abdeckung in diesem Bereich nicht wirklich ausreichend.

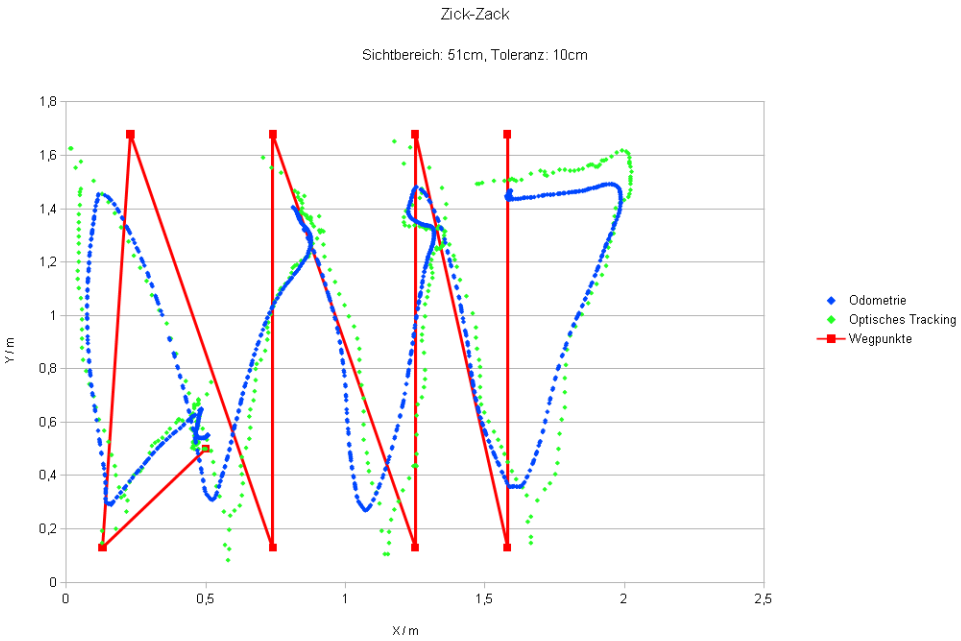


Abbildung 5.20.: Zick-Zack mit Sichtbereich = 51cm, Toleranz = 10cm

In Hinblick auf das optische Tracking, wurde in der Versuchsanordnung aus Abbildung 5.21 der Suchbereich relativ gut abgeflogen. Wenn man jedoch die Odometrie beachtet, merkt man, dass der Quadrokopter teilweise die Wegpunkte nicht ganz abgearbeitet hat.
Da der Quadrokopter teilweise merkwürdige Flugbahnen verfolgt hat, sollte man eher auf einen der anderen Algorithmen zurückgreifen. Die Toleranz sollte im Wesentlichen nicht größer als 0 eingestellt werden, da sonst Wegpunkte anhand der Karte bereits in eine Wand generiert werden würden. Den Sichtbereich kann man je nach Schwerpunkt einstellen. Für ein relativ “sauberes“ Abfliegen der Wegpunkte bietet sich ein größerer Wert an, jedoch kann es dann passieren, dass Bereiche nicht betrachtet werden.

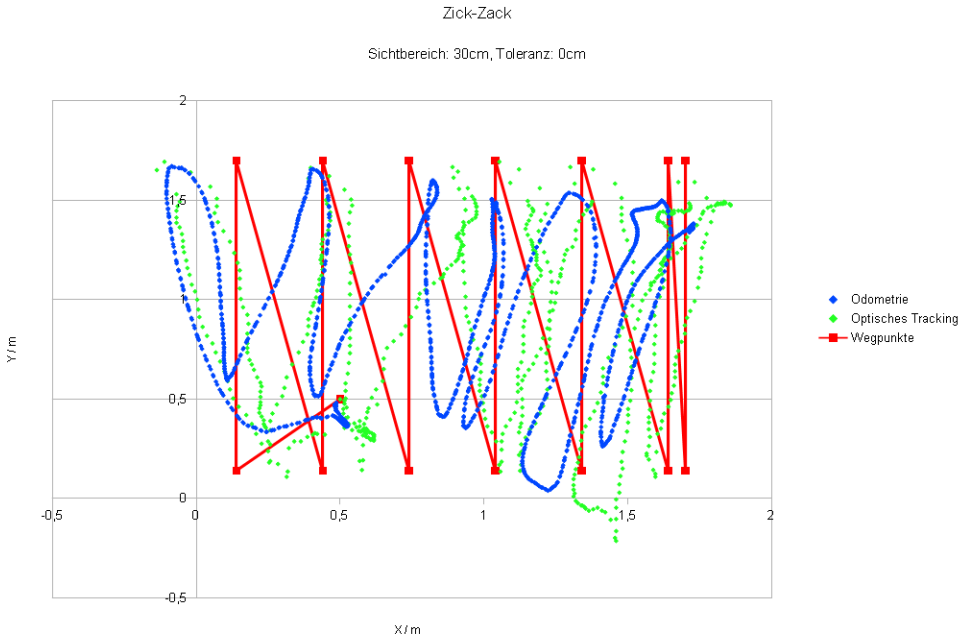


Abbildung 5.21.: Zick-Zack mit Sichtbereich = 30cm, Toleranz = 0cm

5.4. Zusammenfassung

Die Evaluierung der Simulation und der Praxis-Versuche ist allgemein positiv ausgefallen. Die Wegpunkte werden für verschiedene Parameter richtig generiert. Bei der Praxis hat sich gezeigt, dass der Quadrokopter jeweils ein kleines Stück über den Wegpunkt hinaus fliegt, je nachdem wie lang die “Beschleunigungsstrecke“ vom vorherigen Wegpunkt ist. Der Sichtbereich sollte abhängig von den Eigenschaften der Kamera (Öffnungswinkel) und der Flughöhe angepasst werden und besser etwas kleiner als der echte Sichtbereich gewählt werden, da der Quadrokopter sonst unter Umständen manche Bereiche nicht abfliegt. Dabei sollte beachtet werden, dass manche Bereiche mehrfach im Sichtbereich der Kamera sind. Der Toleranz-Parameter ist sinnvoll bei nah beieinander liegenden Wegpunkten zu verwenden, da der Quadrokopter die einzelnen Wegpunkte sonst gar nicht anfliegen würde. Dies ist zum Beispiel bei der Tiefensuche der Fall. Von der allgemeinen Flugbahn jedoch hat sich gezeigt, dass der Quadrokopter bei großer Toleranz das Überschwingen über einen Wegpunkt verstärkt. Um dies zu vermeiden sollte man die Toleranz möglichst klein wählen, beziehungsweise auf Null setzen.

6. Diskussion und Ausblick

6.1. Suche in komplexen Umgebungen

Anhand der Evaluierung der hier implementierten Wegpunktlisten muss in zukünftigen Projekten darauf geachtet werden, dass der Quadrokopter relativ weit von den in der Theorie erstellten Koordinaten abweicht. Bei komplexen Umgebungen mit Wegpunkten, die nah an echten Wänden liegen ist dringend auf die Kollisionserkennung zu achten. Das Erstellen der Wegpunkte für nicht triviale Umgebungen sollte darauf ausgelegt sein, Wegpunkte möglichst über lange gerade Strecken zu legen, und dann an den Quadrokopter jeweils nur den letzten Wegpunkt der Gerade zu übergeben. Komplizierte Wendemanöver sollten vermieden werden. An der Auswertung der Zick-Zack-Bewegung sieht man besonders gut, dass der Quadrokopter bei einer Drehung um fast 180 ° ziemlich merkwürdige Flugbahnen verfolgt. (siehe Abbildung 6.1)

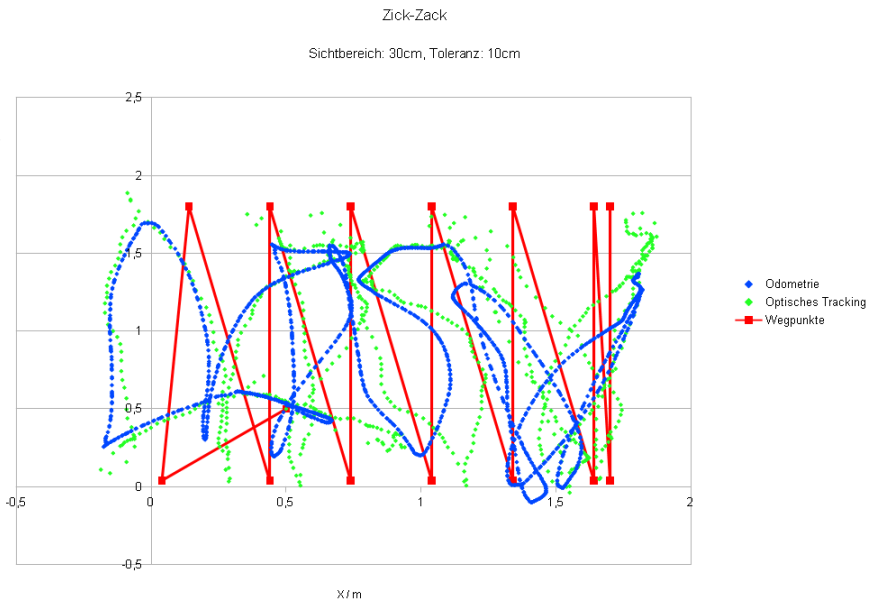


Abbildung 6.1.: Zick-Zack: merkwürdige Flugbahn

6.2. Kombination von Suchalgorithmen mit Objekterkennung und Kollisionserkennung

Das eigentliche Ziel, das Finden eines Objekts, wurde im Rahmen dieser Arbeit nicht getestet. Dies ist einer der nächsten Schritte für das Projekt Lebensretter mit Propellern. Es muss darauf geachtet werden, dass der Quadrokopter durch Schwankungen beziehungsweise durch die generierten Wegpunkte über den selben Ort mehrfach fliegen kann. Zusätzlich muss eine Rückkopplung zur Kollisionserkennung mit den Wegpunkten eingerichtet werden. Das heißt wenn der Quadrokopter durch die Kollisionserkennung merkt, dass er zu nahe an der Wand ist, müssten die Wegpunkte um einen gewissen Wert zurückgesetzt werden.

6.3. Feedback über bereits abgesuchte Bereiche

Da die tatsächliche Bewegung des Quadrokopters weit von der in der simulierten Bewegung abweicht, müsste man eine Korrektur beziehungsweise eine Rückkopplung vom Quadrokopter während des Fluges berechnen lassen. Dies wäre zum Beispiel durch eine Regelung der Odometrie-Werte mit den Verbindungsstrecken der Wegpunkte umsetzbar. Beziehungsweise muss anhand der Odometrie gemerkt werden, wo der Quadrokopter sein sollte, jedoch bisher noch nicht war. Diese Bereiche müssten neben den restlichen Wegpunkten noch abgesucht werden.

6.4. Simulation von Heuristischen Suchen

Anhand der hier implementierten GUI kann sehr einfach eine komplexe Umgebung erstellt werden. Es dient als Grundlage für die Simulation beliebiger Suchalgorithmen. In dieser Arbeit wurde der Fokus auf uninformierte Suchen gelegt, jedoch können auch heuristische Suchen in dieser Umgebung umgesetzt werden. Hierbei muss dringend darauf geachtet werden, dass die Wegpunkte in genügendem Abstand zu Wänden erstellt werden, da der Quadrokopter in der Realität teilweise weit von den generierten Wegpunkten abweicht.

6.5. Anpassen von iterativ generierten Wegpunkten

Da in der hier verwendeten Version des Quadrokopters, die Wegpunkte bereits 20cm vor dem eigentlichen Erreichen, diese als erreicht vermerkt werden, muss man für beliebige iterative Suchalgorithmen versuchen mehrere Wegpunkte zusammenzufassen. Man kann die Wegpunkte, welche auf geraden Strecken hintereinander liegen, über einen einzigen, den letzten Wegpunkt an den Quadrokopter übergeben. Auch für versetzte Wegpunkte die auf einer Kurve liegen, kann unter der Berücksichtigung, dass der Quadrokopter über manche Wegpunkte hinaus fliegt, eine ähnliche Zusammenfassung genutzt werden.

7. Literaturverzeichnis

[Bishop 2010] BISHOP, Jacob: *Search Pattern Generation and Path Management for Search over Rough Terrain with a Small UAV: Master Thesis*. 2010

[Coleman et al. 2012] COLEMAN, Catherine ; FUNK, Joseph ; SALVATI, James ; WHIPPLE, Christopher: *Design of an Autonomous Platform for Search and Rescue UAV Networks*. 2012

[Görz 2003] GÖRZ, Günther: *Handbuch der künstlichen Intelligenz*. 4. München and Wien : Oldenbourg, 2003. – ISBN 9783486272123

[Grzonka et al. 2009] GRZONKA, Slawomir ; GRISETTI, Giorgio ; BURGARD, Wolfram: *Towards a Navigation System for Autonomous Indoor Flying*. 2009

[Hart et al. 1968] HART, P. ; NILSSON, N. ; RAPHAEL, B.: A formal basis for the heuristic determination of minimum cost paths. In: *Aus: IEEE Transactions Of Systems Science And Cybernetics, Vol Ssc-4(1968),2 ,S.100-107* (1968)

[Hopcroft und Ullman 1979] HOPCROFT, John E. ; ULLMAN, Jeffrey D.: *Introduction to automata theory, languages, and computation*. Reading and Mass : Addison-Wesley, 1979 (Addison-Wesley series in computer science). – ISBN 9780201029888

[Kleiner und Nebel 2013] KLEINER, Alexander ; NEBEL, Bernhard: *Search Algorithms and Pathfinding*. 2013. – URL `http://www.informatik.uni-freiburg.de/~ki/teaching/ws1011/imap/04_SearchAlgorithmsAndPathPlanningMAS_PartA.pdf`

[LaValle 2006] LAVALLE, Steven M.: *Planning algorithms*. Cambridge and New York : Cambridge University Press, 2006. – ISBN 9780521862059

[Lunze 1994] LUNZE, Jan: *Kuenstliche Intelligenz fuer Ingenieure: Methodische Grundlagen und Softwaretechnologie*. München : Oldenbourg, 1994 (Methodische Grundlagen und Softwaretechnologie). – ISBN 9783486222876

[Patnaik 2006] PATNAIK, Srikanta: *Studies in computational intelligence*. Bd. 8: *Innovations in robot mobility and control*. Berlin and New York : Springer, 2006. – ISBN 9783540268925

[Sarid und Shapiro 2009] SARID, Shahar ; SHAPIRO, Amir: *Classifying the multi robot path finding problem into a quadratic competitive complexity class*. 2009

[Schmitt 2012] SCHMITT, Norbert: *Intelligentes Mapping für Indoor-Quadrocopter: Bachelor Thesis*. 2012

[Siegwart et al. 2011] SIEGWART, Roland ; NOURBAKHSH, Illah R. ; SCARAMUZZA, Davide: *Introduction to autonomous mobile robots*. 2. Cambridge and MA [u.a.] : MIT Press, 2011 (Intelligent robotics and autonomous agents series). – ISBN 0262015358

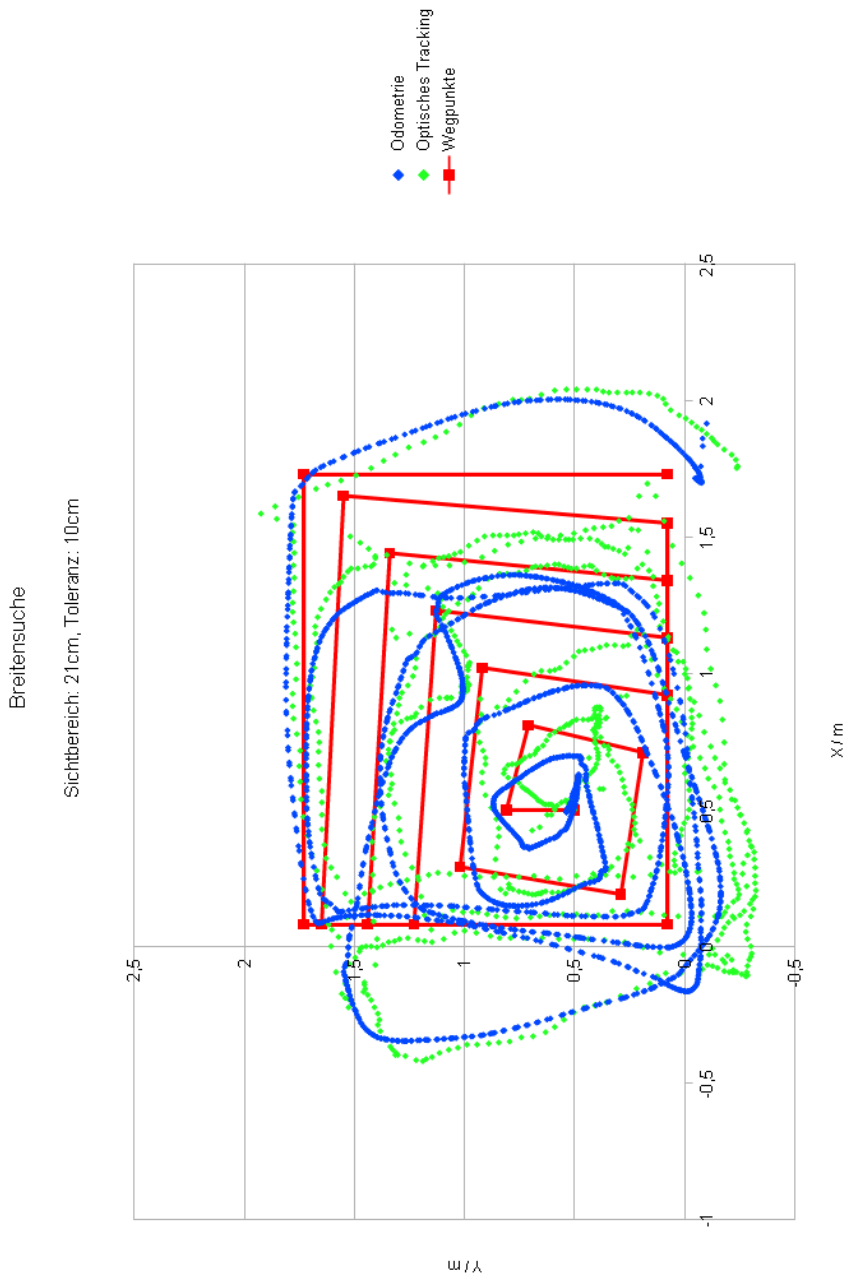
[Sommer 2008] SOMMER, Ulli: *Roboter selbst bauen: Das grosse Praxisbuch für Einsteiger und Fortgeschrittene*. Poing : Franzis, 2008 (Franzis Experimente). – ISBN 3772341098

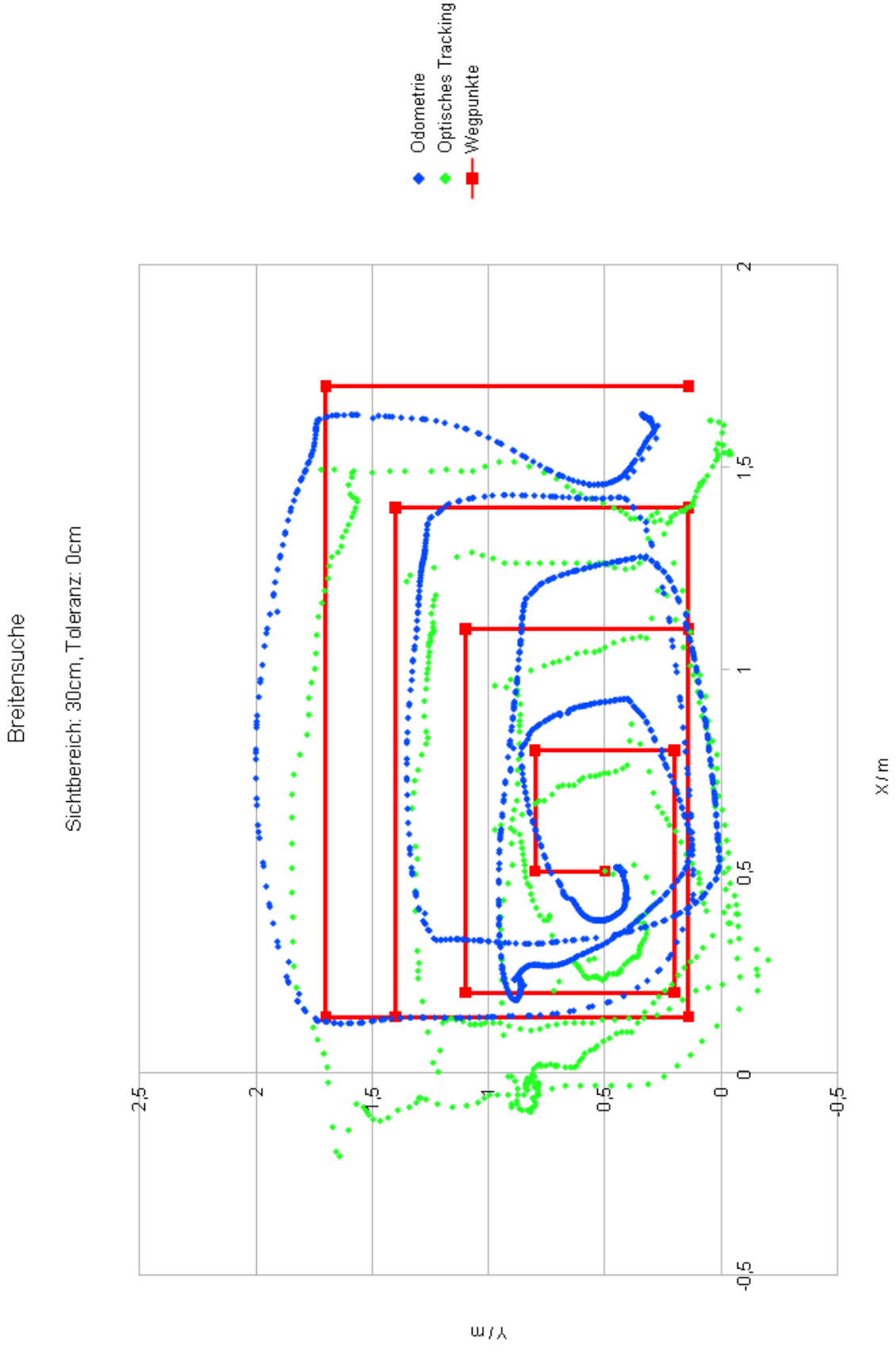
[Soundararaj et al. 2009] SOUNDARARAJ, Sai P. ; SUJEETH, Arvind ; SAXENA, Ashutosh: *Autonomous Indoor Helicopter Flight using a Single Onboard Camera*. 2009

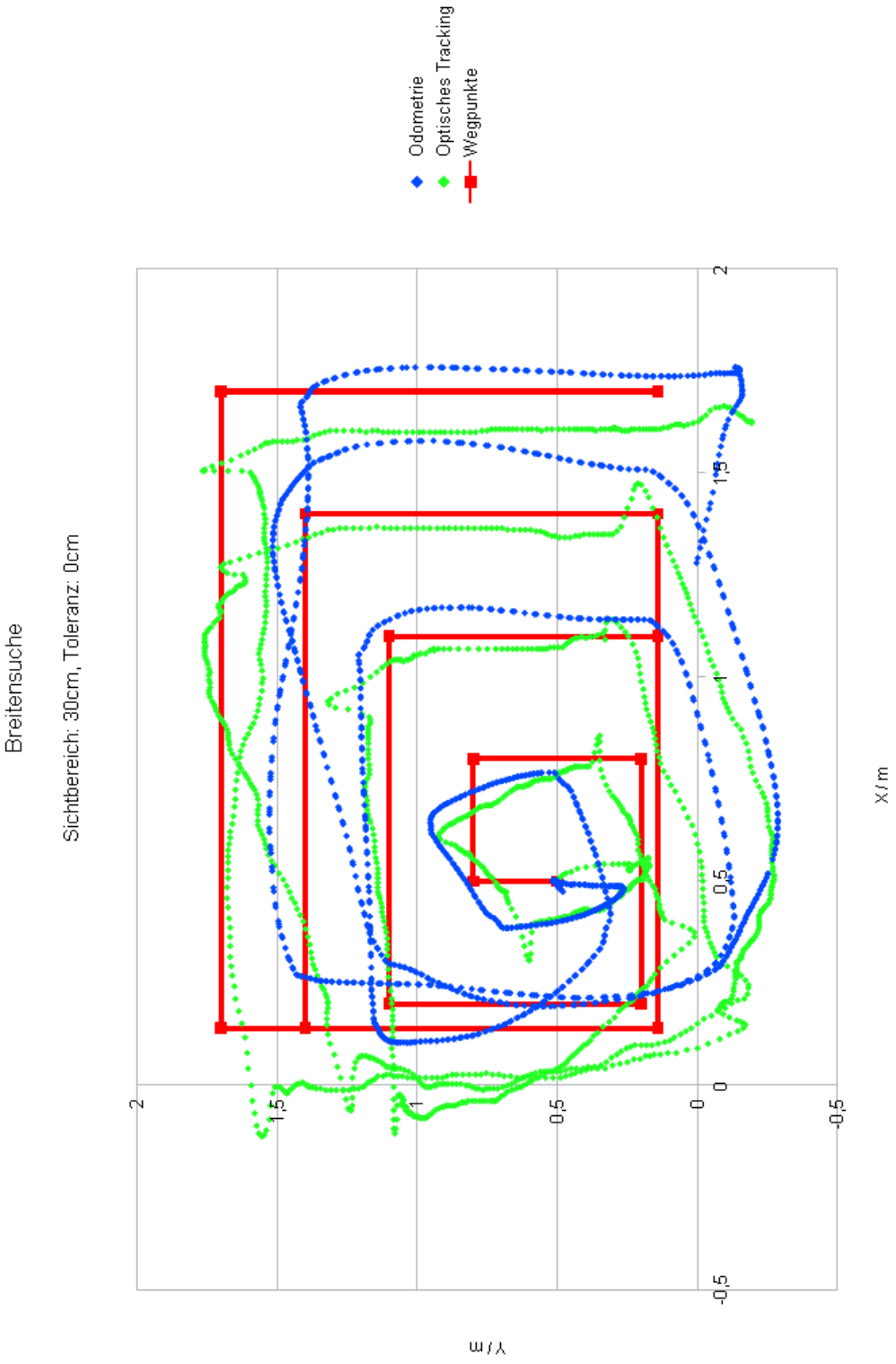
[Strohmeier 2012] STROHMEIER, Michael: *Implementierung und Evaluierung einer Positionsregelung unter Verwendung des optischen Flusses: Bachelor Thesis*. 2012

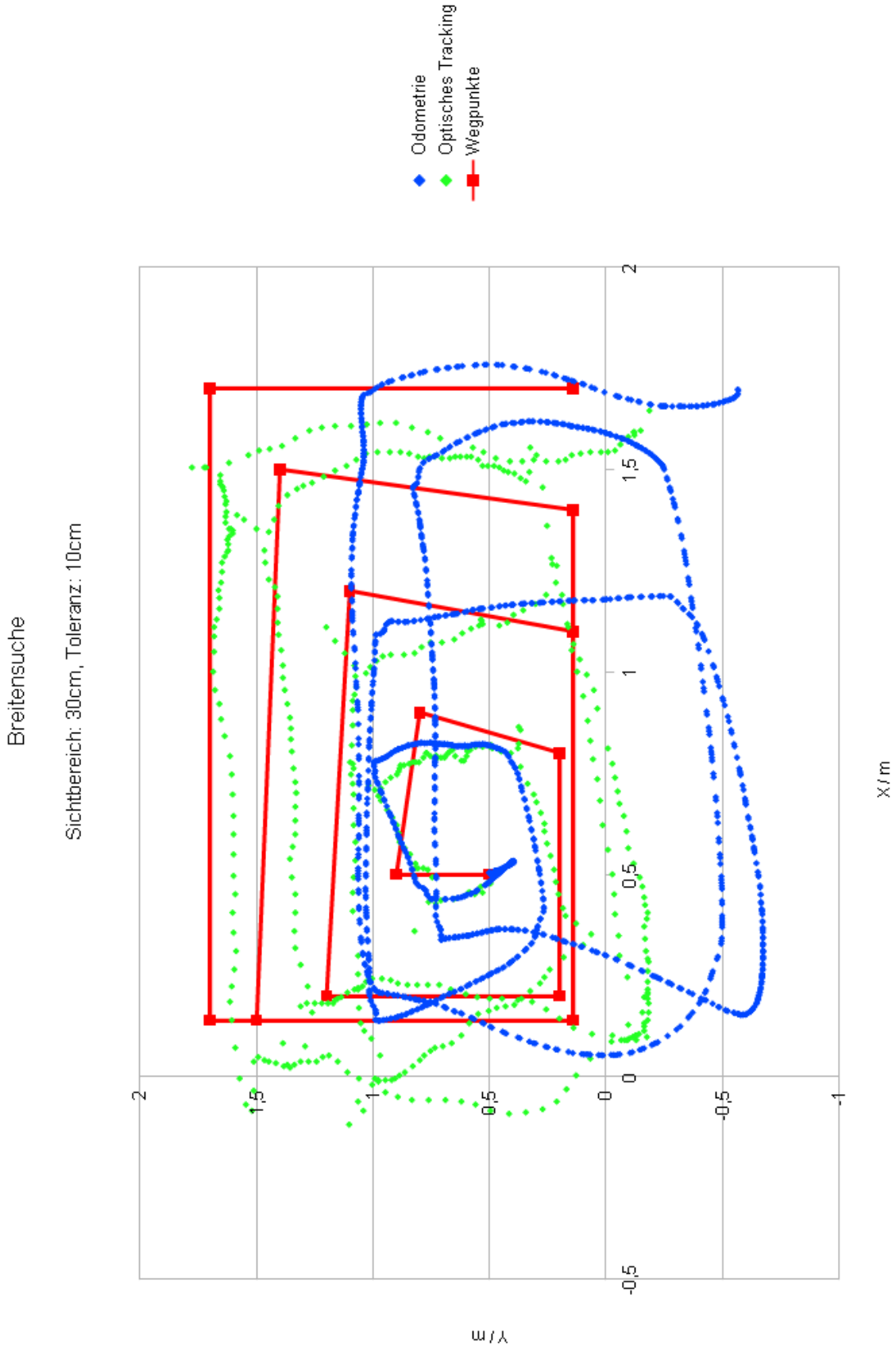
A. Anhang

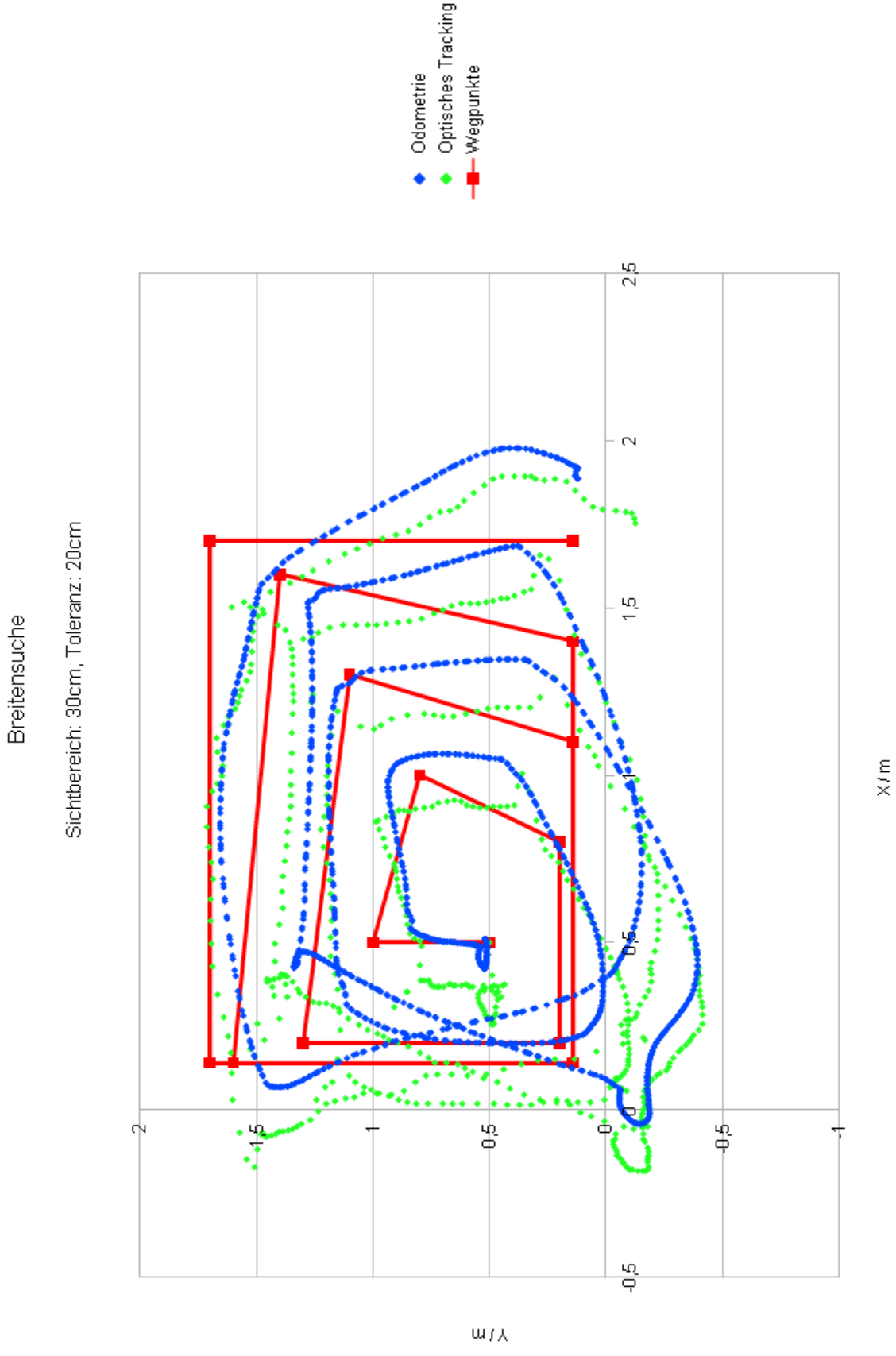
Bilder zur Evaluierung der Praxis:

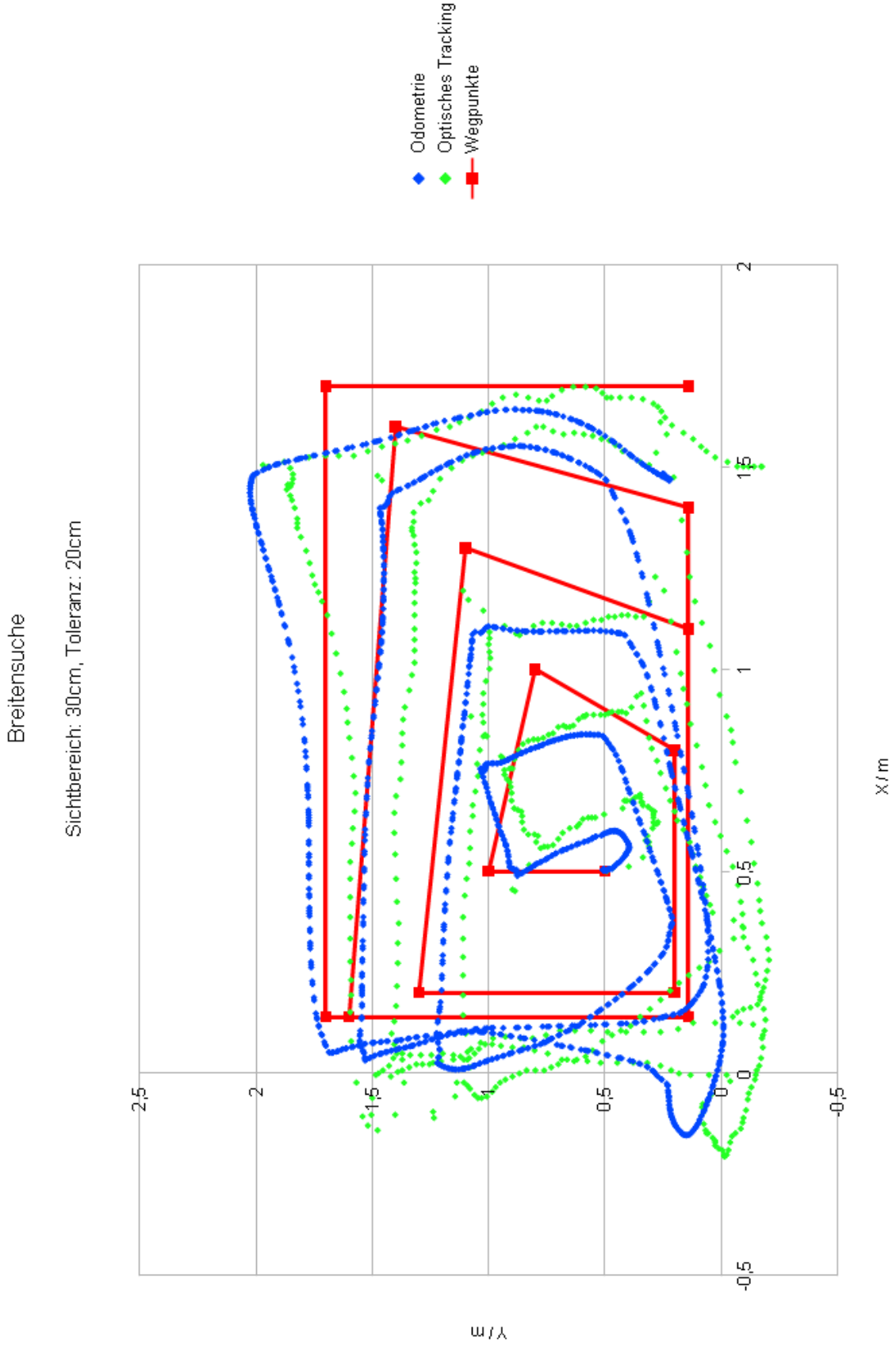


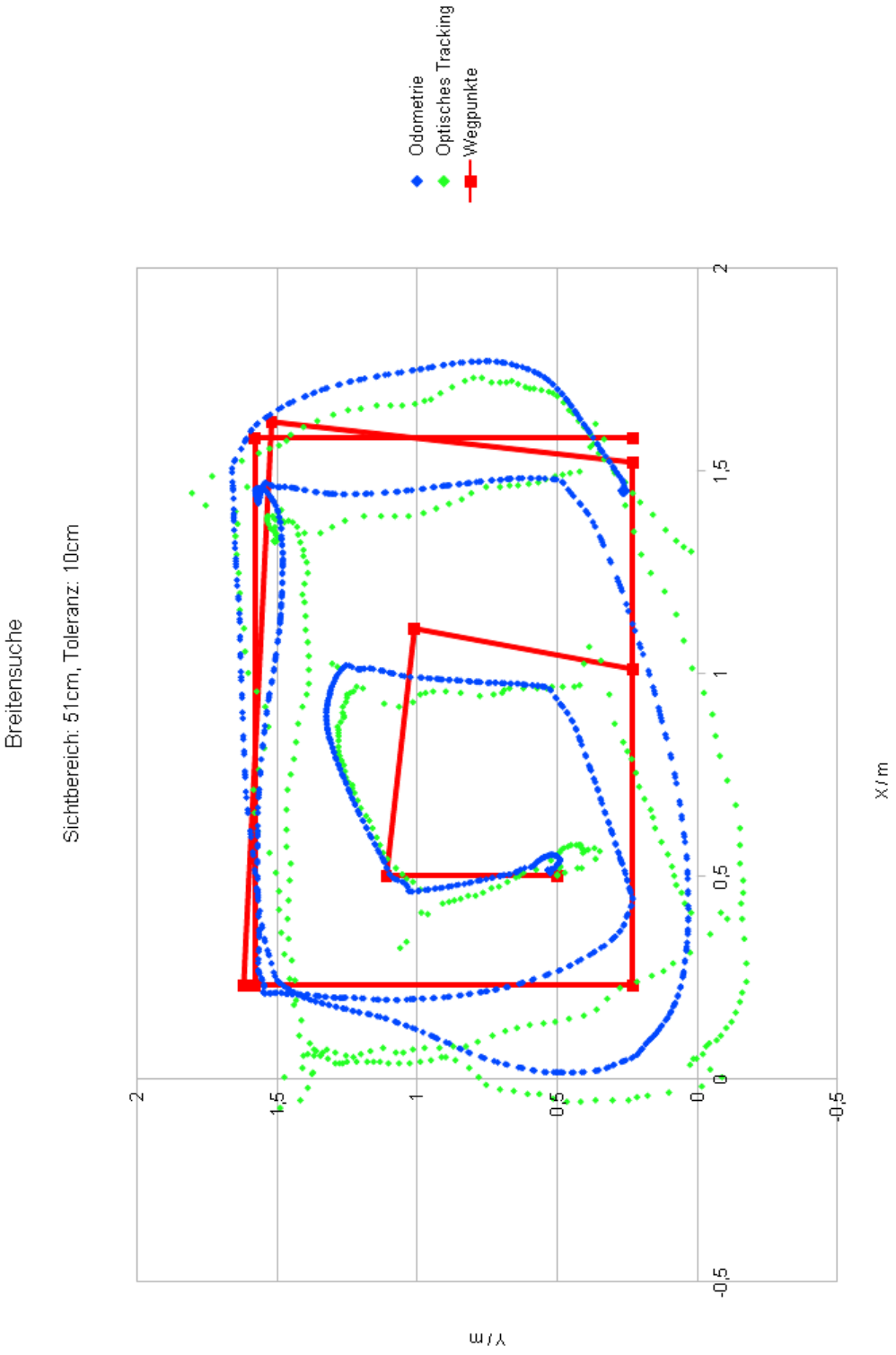

Odometrie
Optisches Tracking
Wegpunkte
Breitensuche
Sichtbereich: 30cm, Toleranz: 0cm
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5

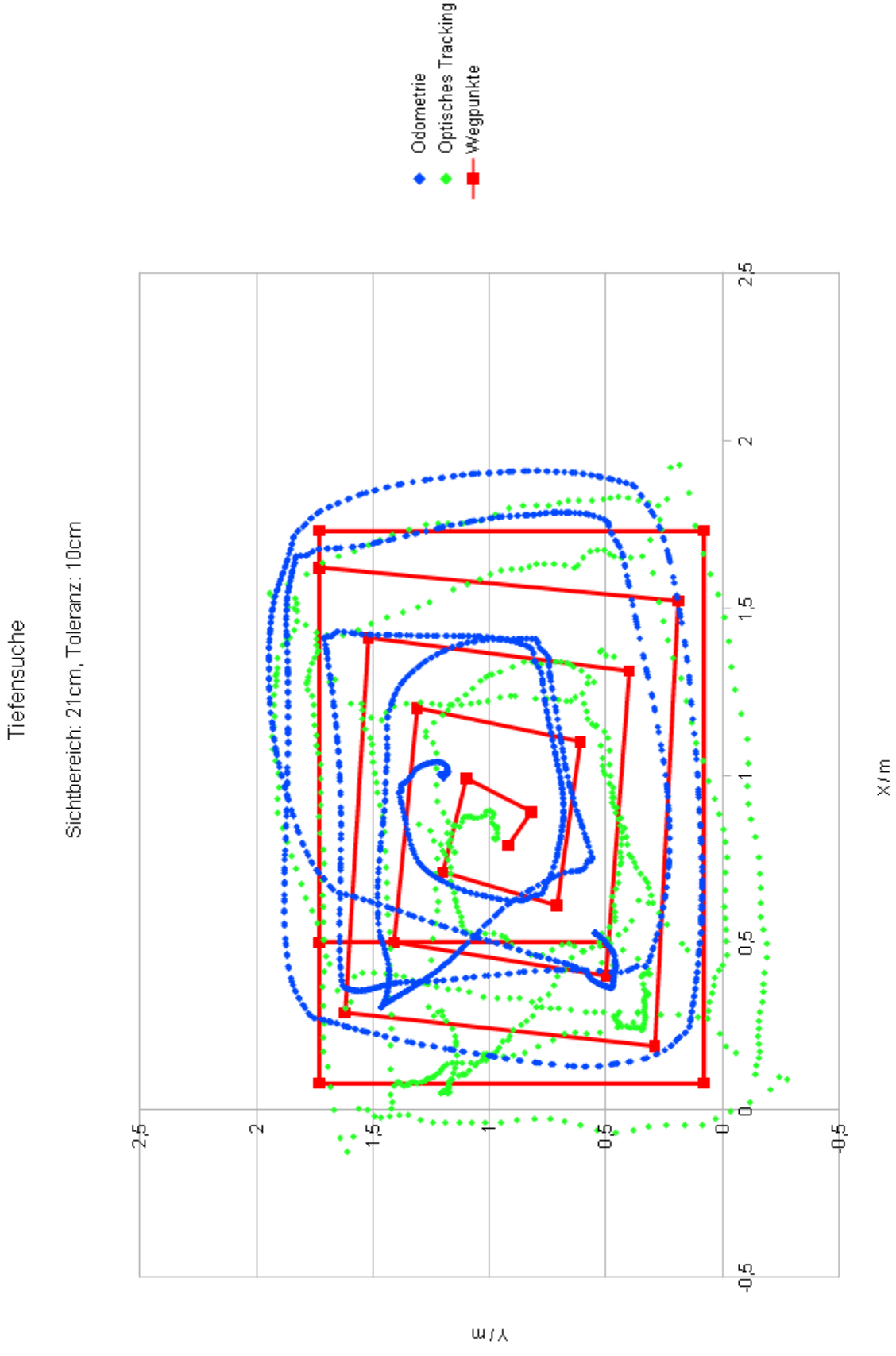

Odometrie
Optisches Tracking
Wegpunkte
Breitensuche
Sichtbereich: 30cm, Toleranz: 0cm
Y / m
X / m
2
1,5
1
0,5
0
-0,5

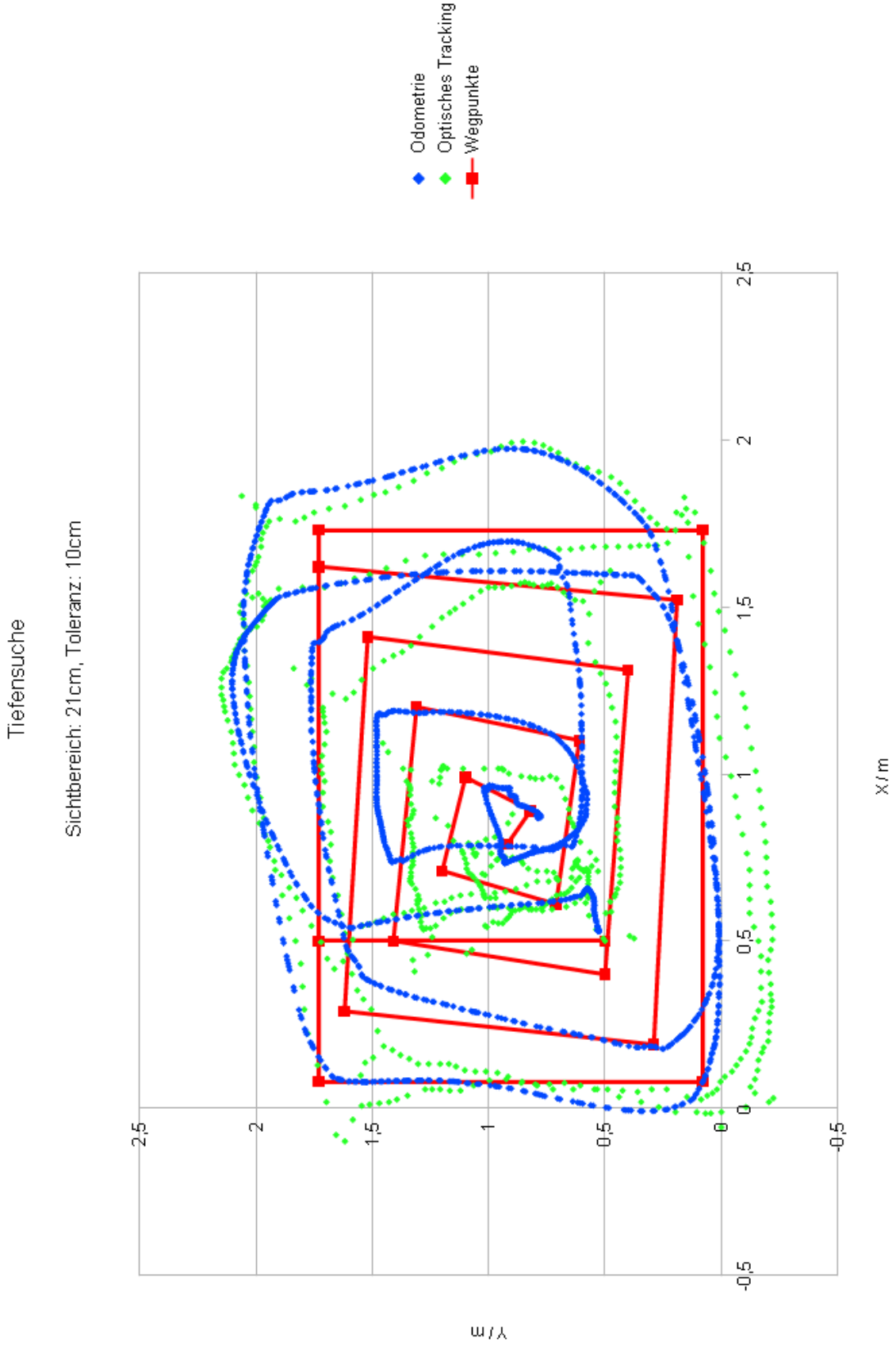

Odometrie
Optisches Tracking
Wegpunkte
Breitensuche
Sichtbereich: 30cm, Toleranz: 10cm
X / m
Y / m
2
1,5
1
0,5
0
-0,5
-1

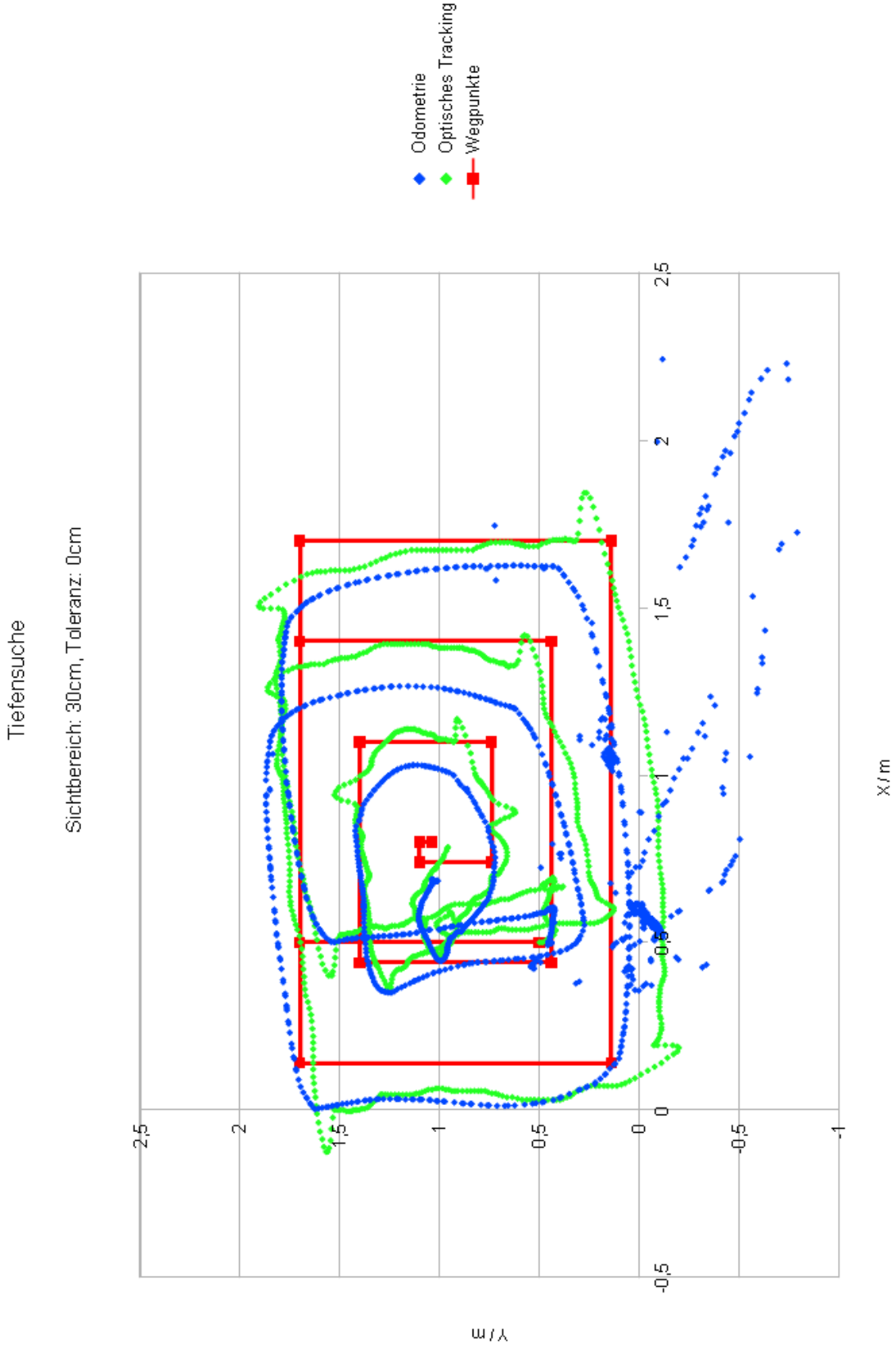

Odometrie
Optisches Tracking
Wegpunkte
Breitensuche
Sichtbereich: 30cm, Toleranz: 20cm
X / m
Y / m
2,5
2
1,5
1
0,5
0
-0,5
-1
2
1,5
1
0,5
0
-0,5

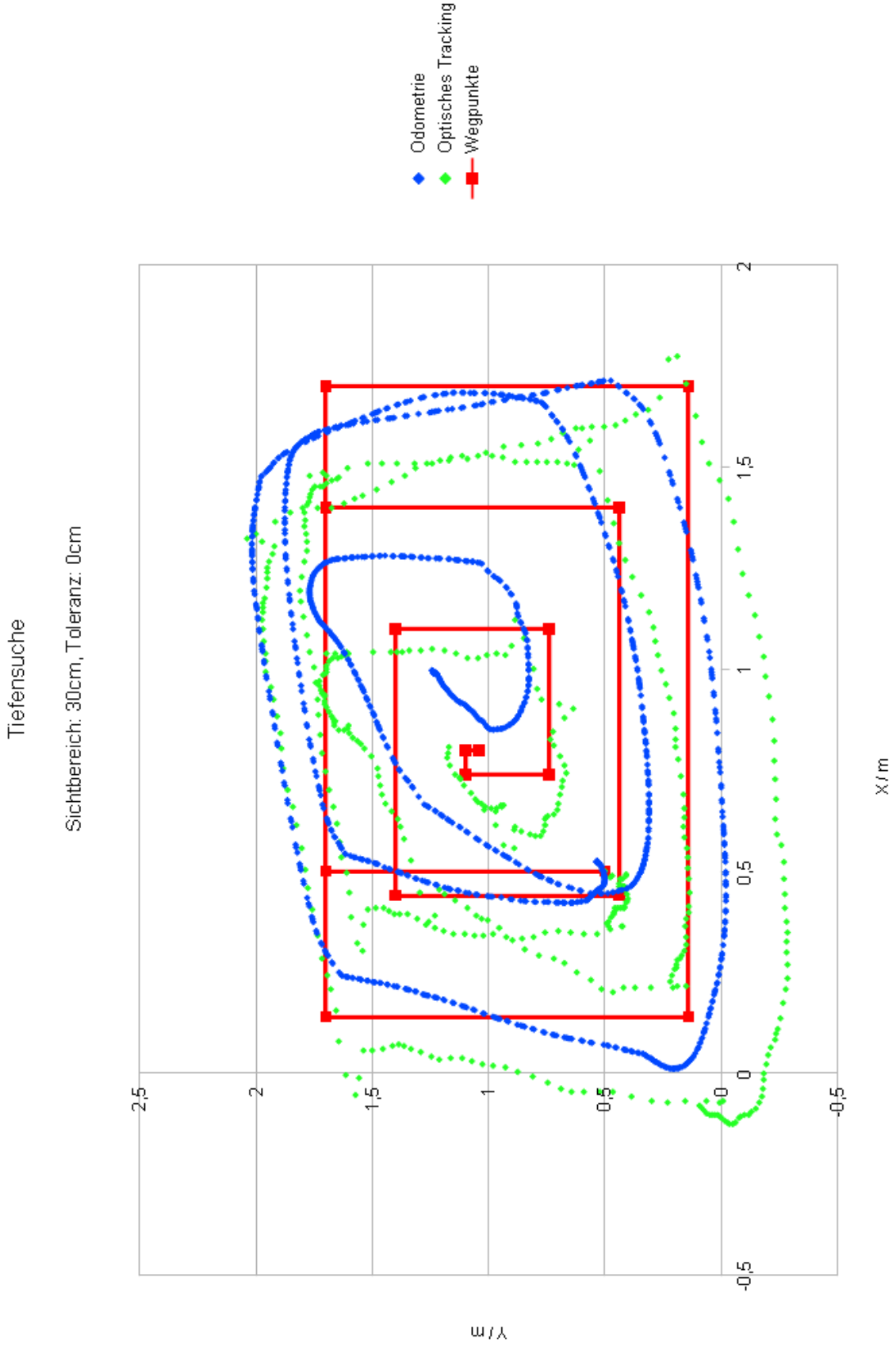

Odometrie
Optisches Tracking
Wegpunkte
Breitensuche
Sichtbereich: 30cm, Toleranz: 20cm
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5

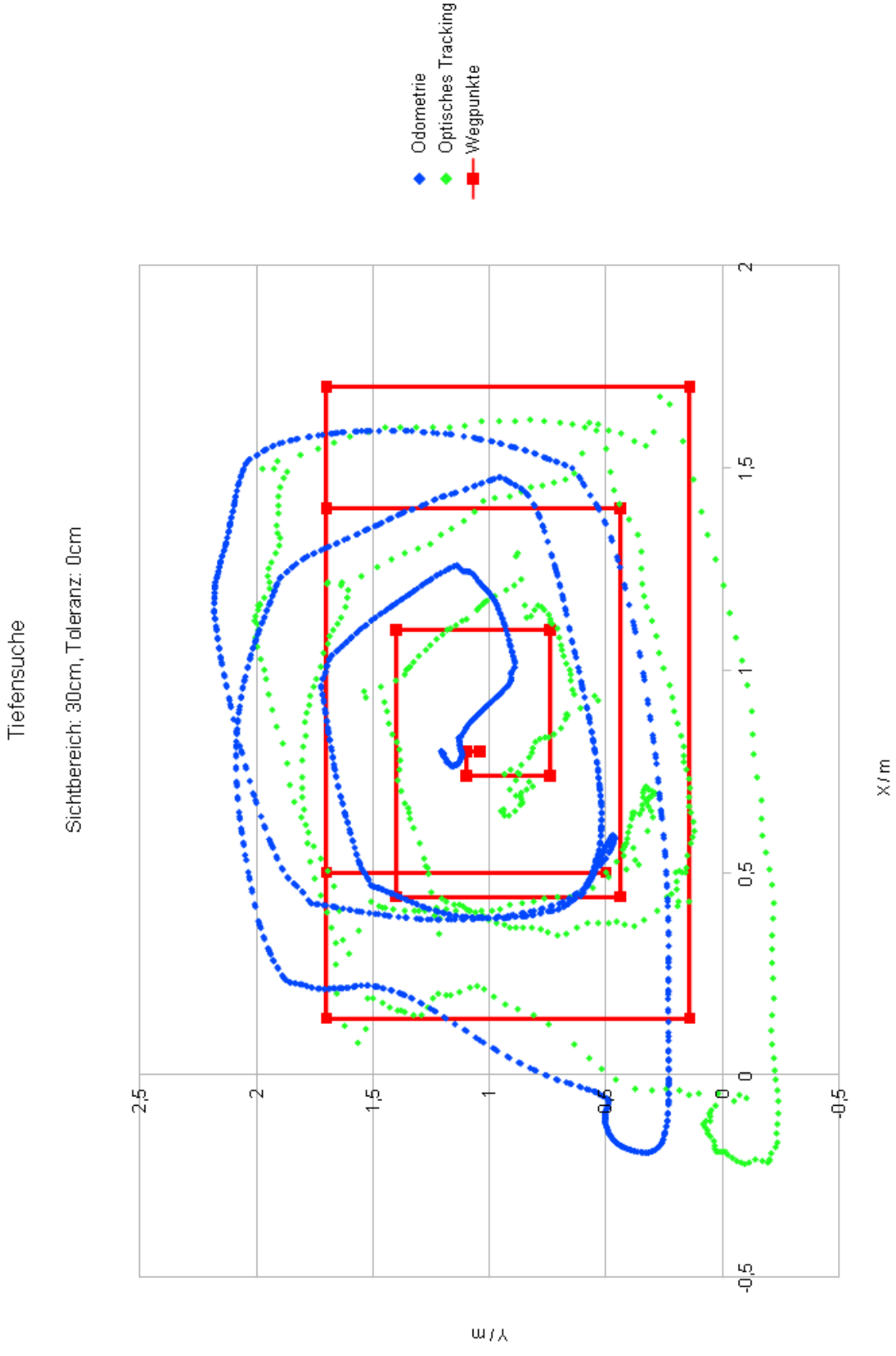

Odometrie
Optisches Tracking
Wegpunkte
Breitensuche
Sichtbereich: 51cm, Toleranz: 10cm
Y / m
X / m
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5

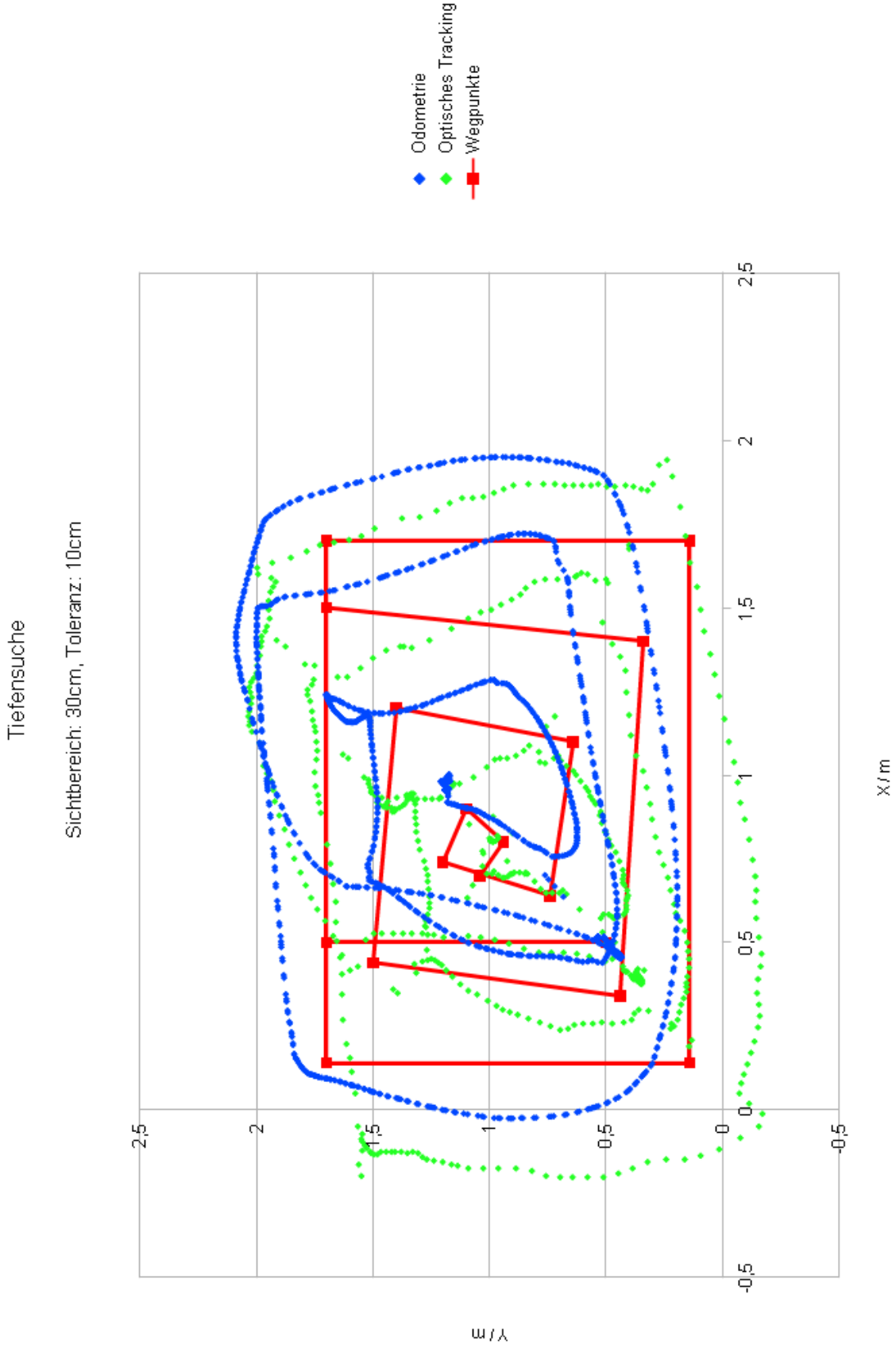

Tiefensuche
Sichtbereich: 21cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5

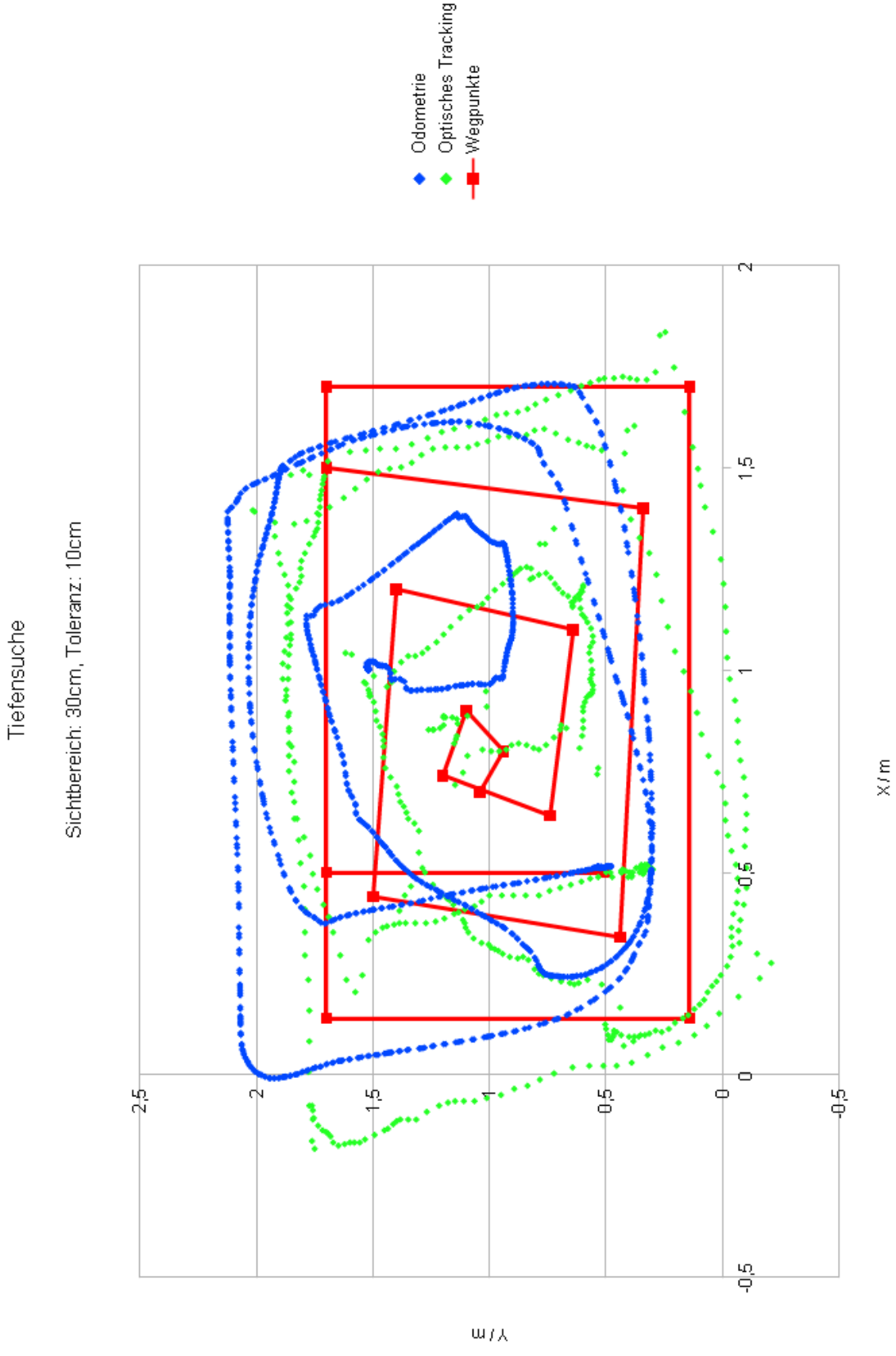

Tiefensuche
Sichtbereich: 21cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5

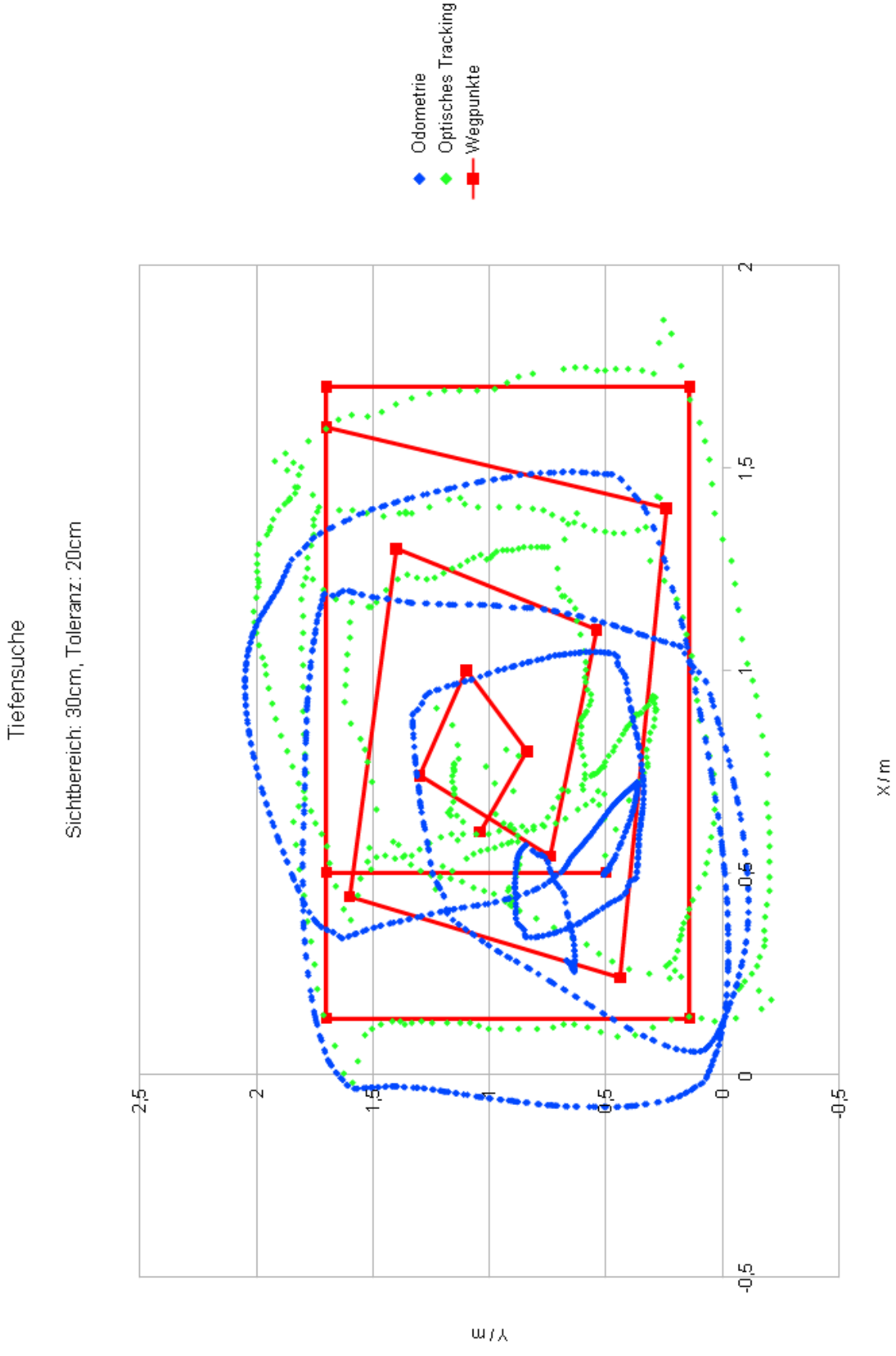

Tiefensuche
Sichtbereich: 30cm, Toleranz: 0cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
-1

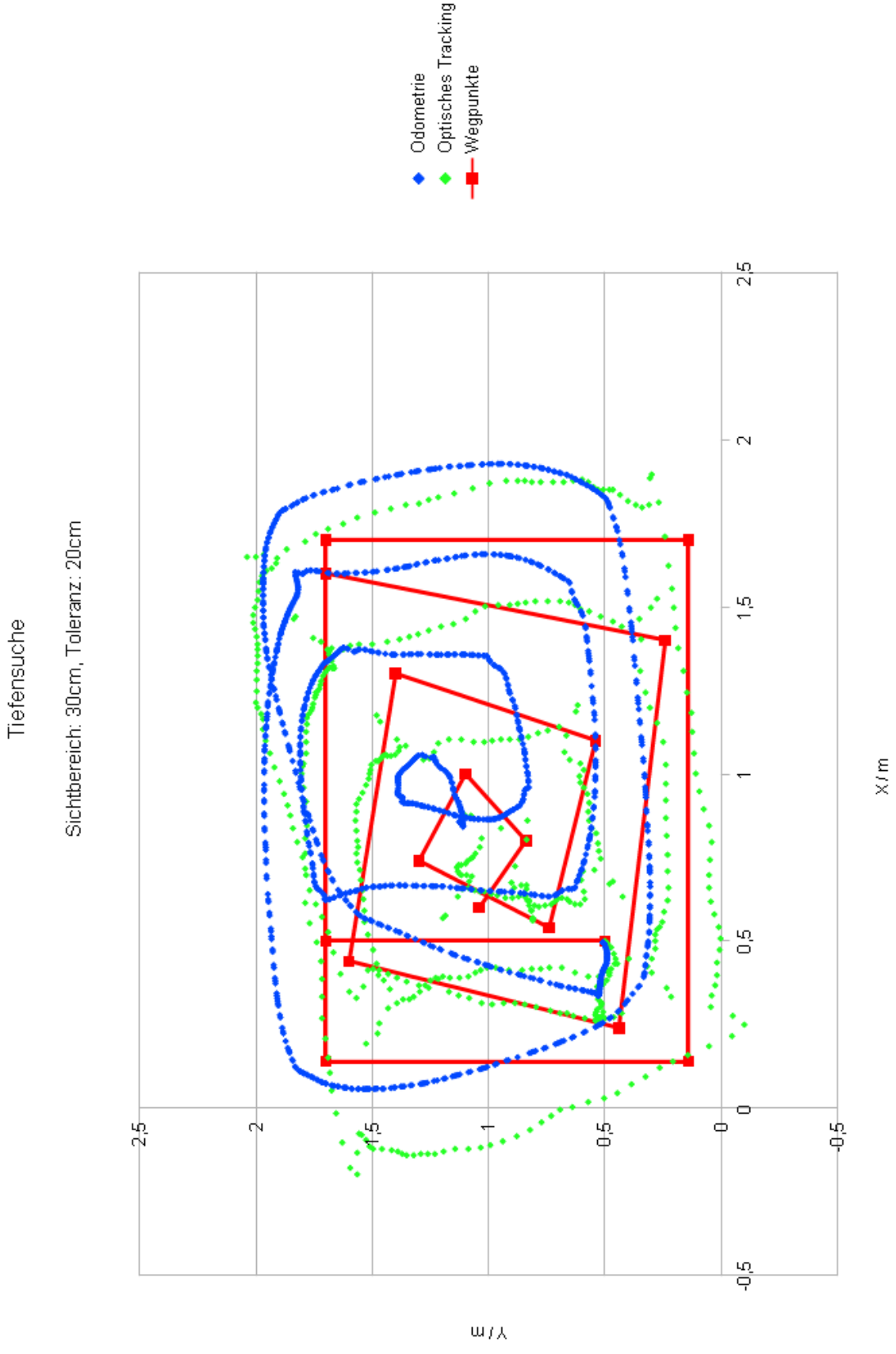

Tiefensuche
Sichtbereich: 30cm, Toleranz: 0cm
Odometrie
Optisches Tracking
Wegpunkte
2,5
2
1,5
1
0,5
0
-0,5
Y / m
2
1,5
1
0,5
0
-0,5
X / m

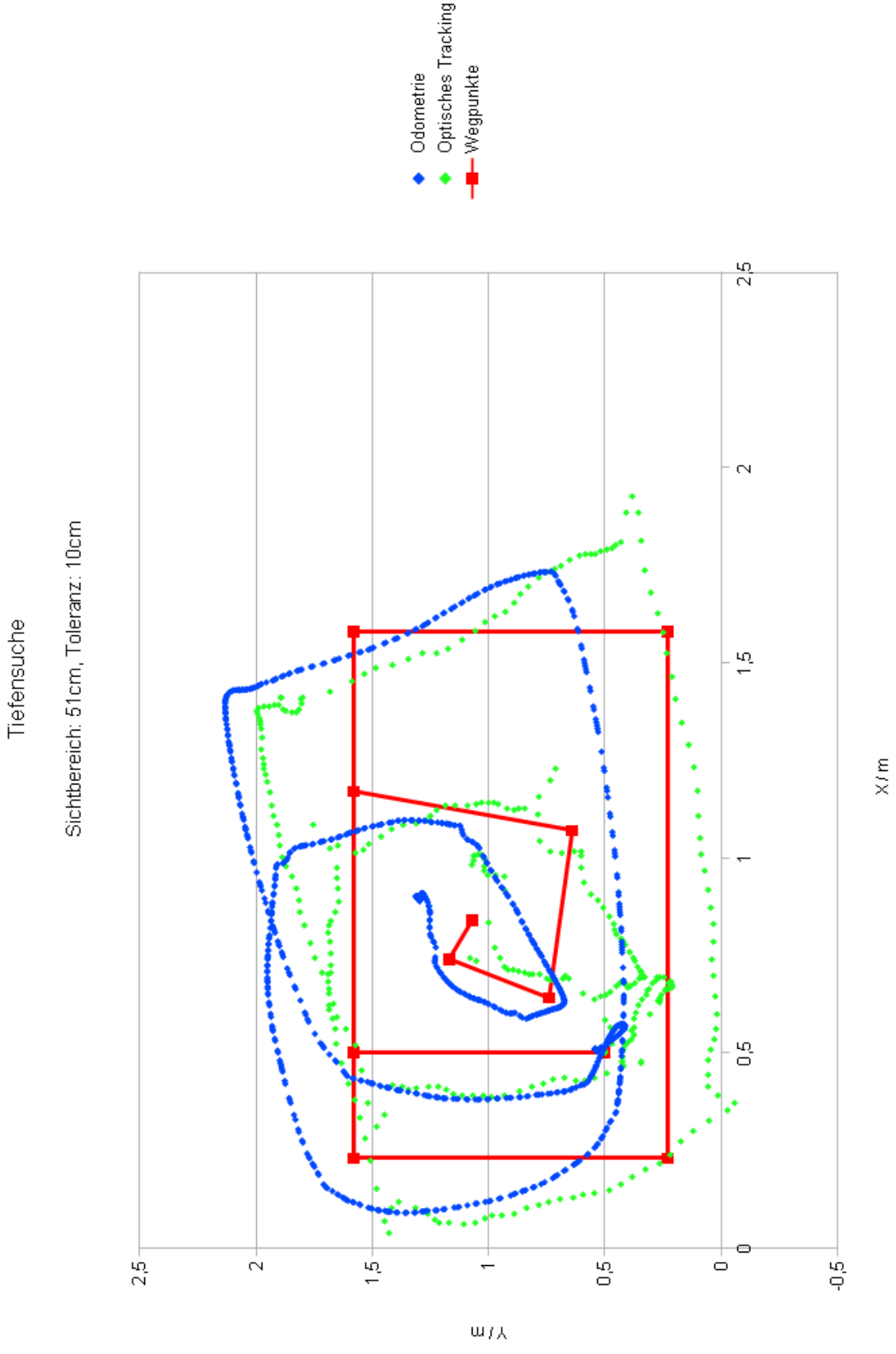

Tiefensuche
Sichtbereich: 30cm, Toleranz: 0cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
-0,5
0
0,5
1
1,5
2

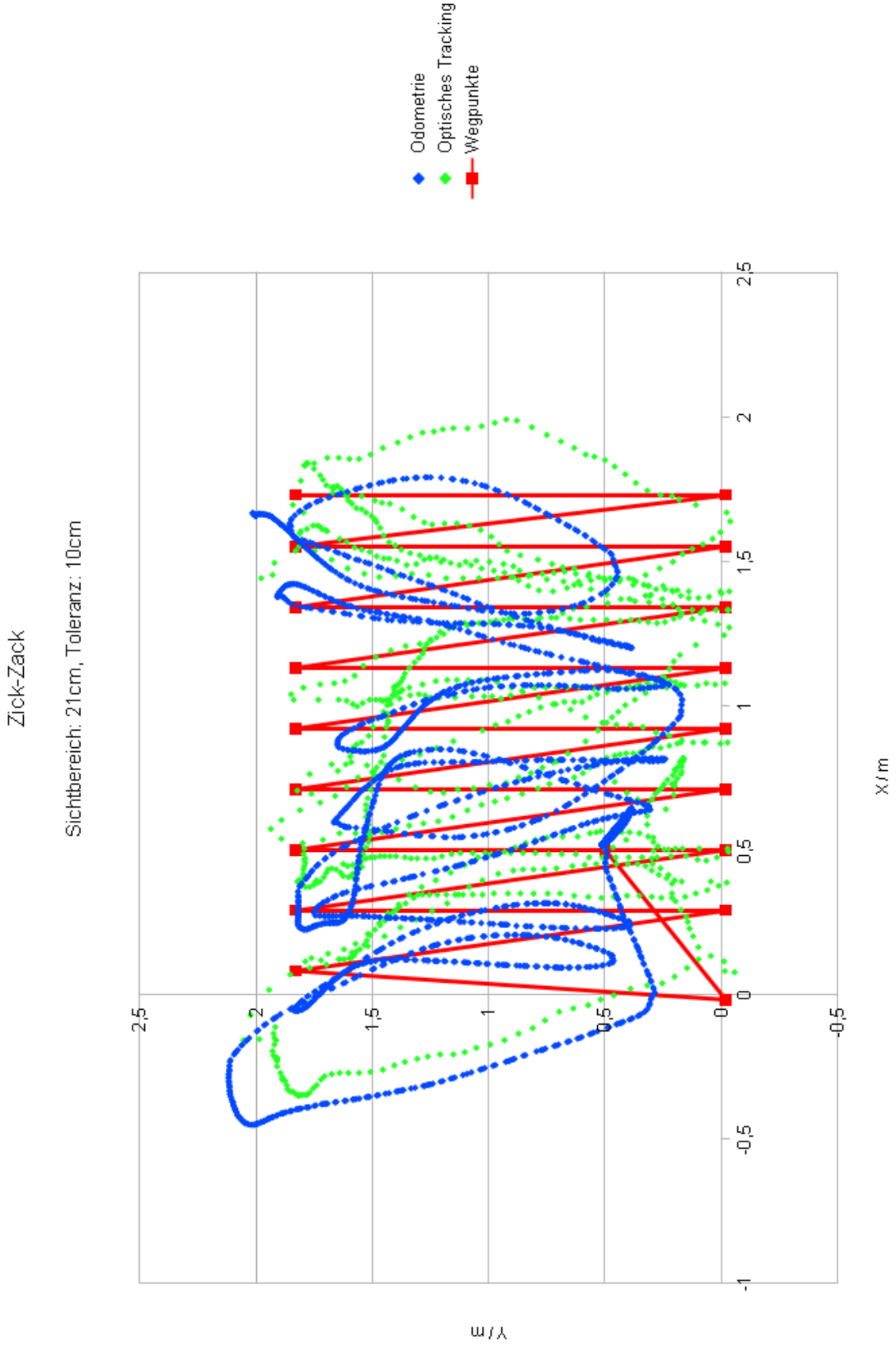

Tiefensuche
Sichtbereich: 30cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5

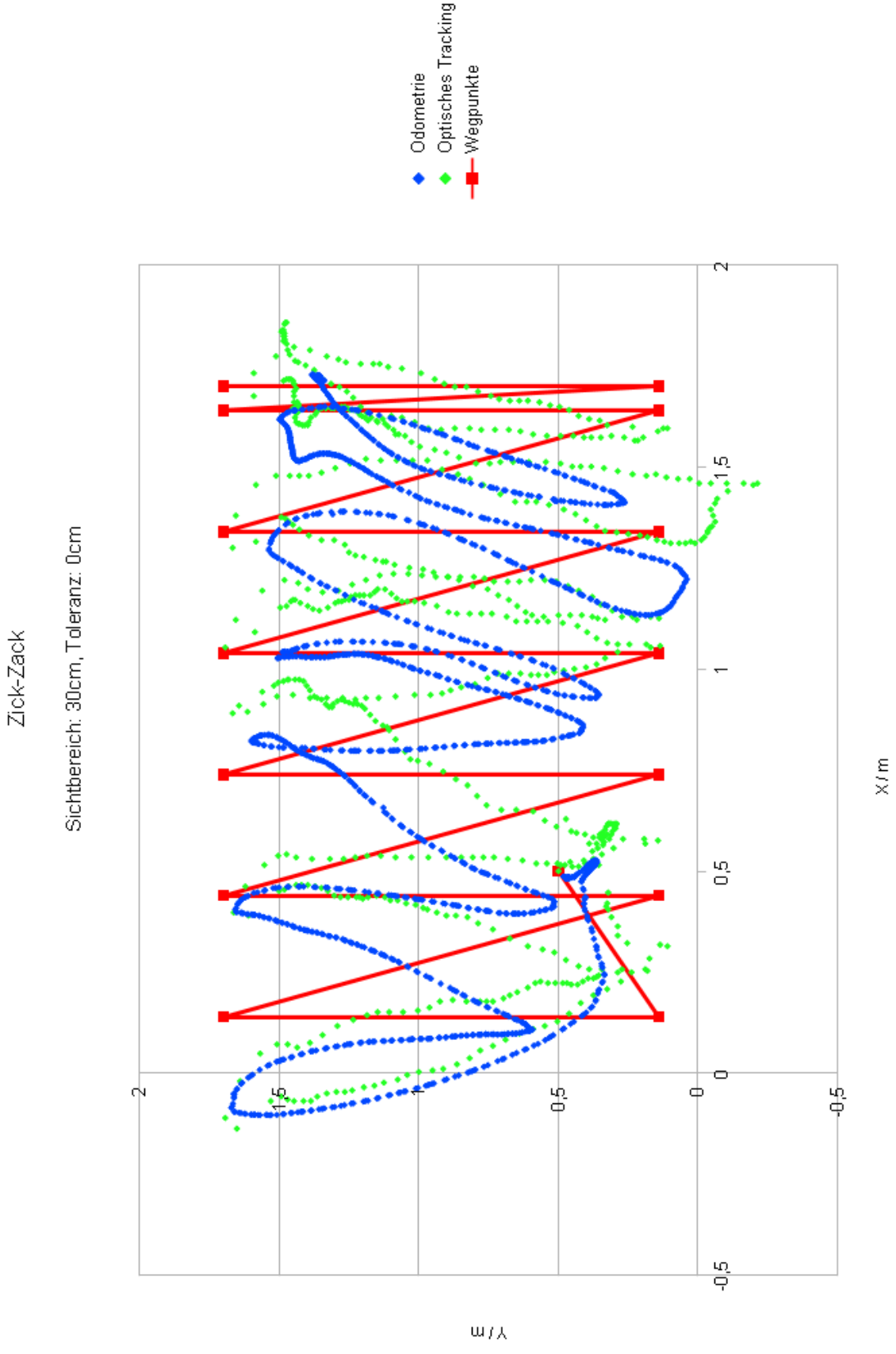

Tiefensuche
Sichtbereich: 30cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5

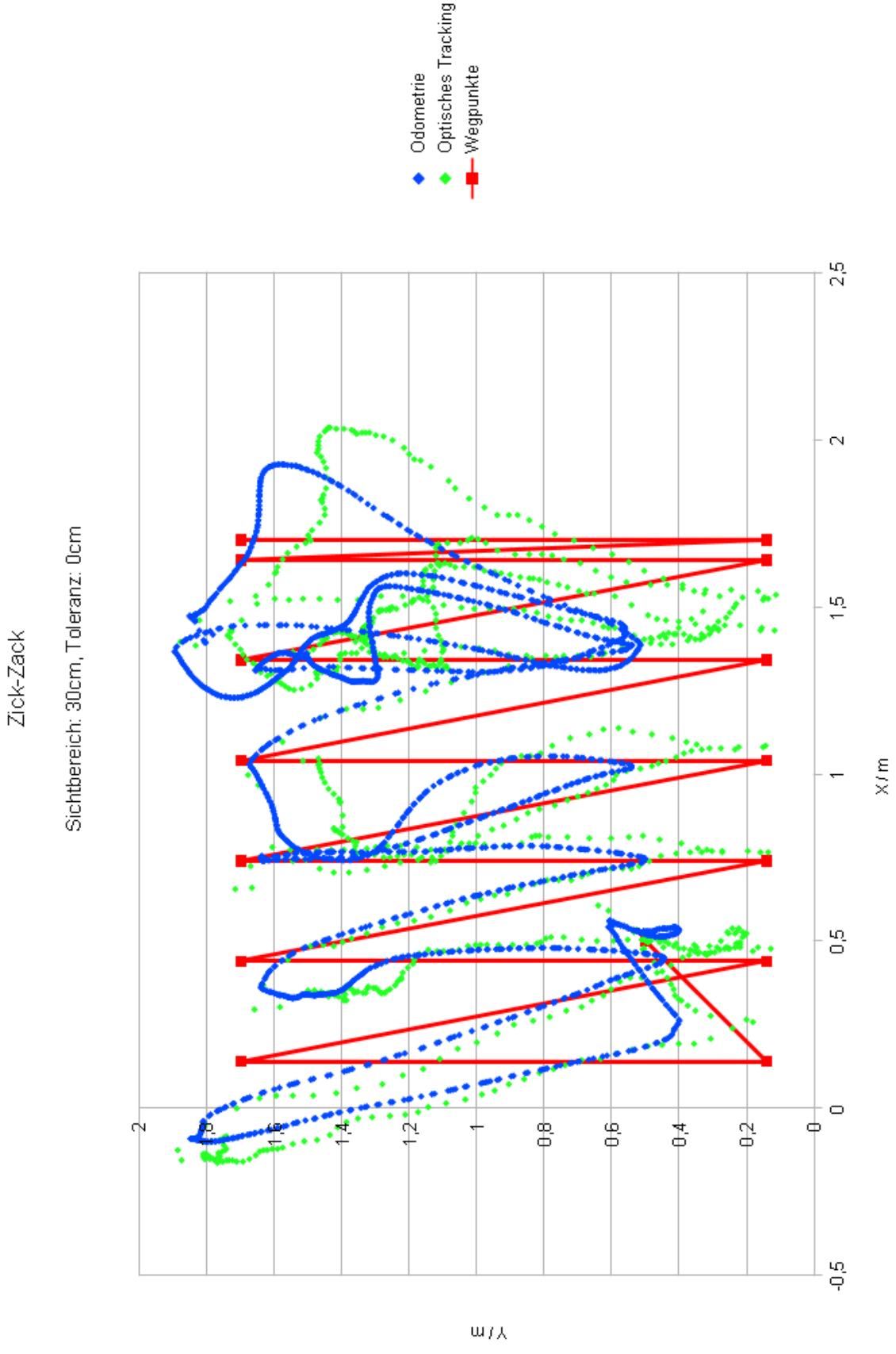

Tiefensuche
Sichtbereich: 30cm, Toleranz: 20cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5

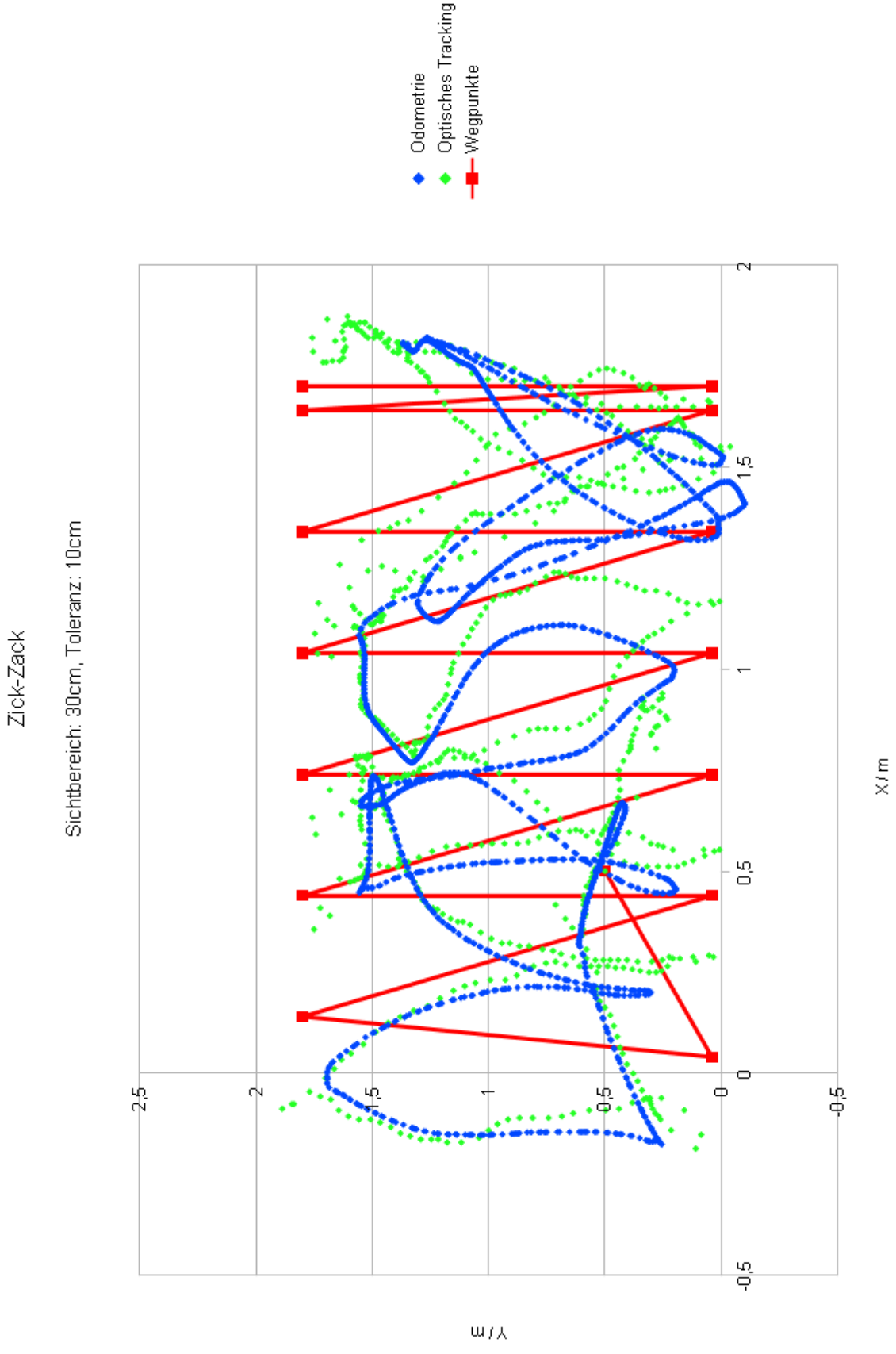

Odometrie
Optisches Tracking
Wegpunkte
Tiefensuche
Sichtbereich: 30cm, Toleranz: 20cm
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5

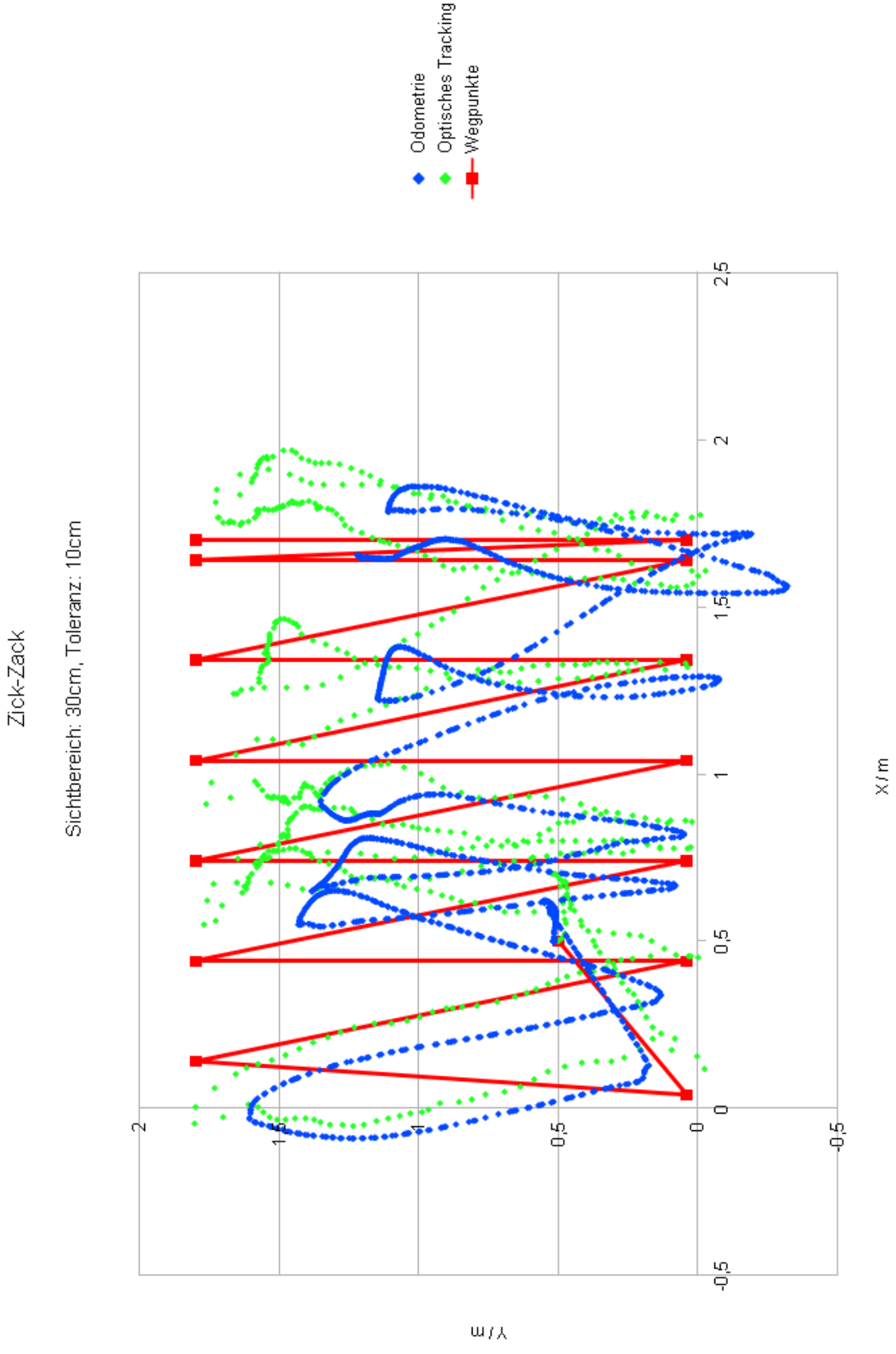

Tiefensuche
Sichtbereich: 51cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5

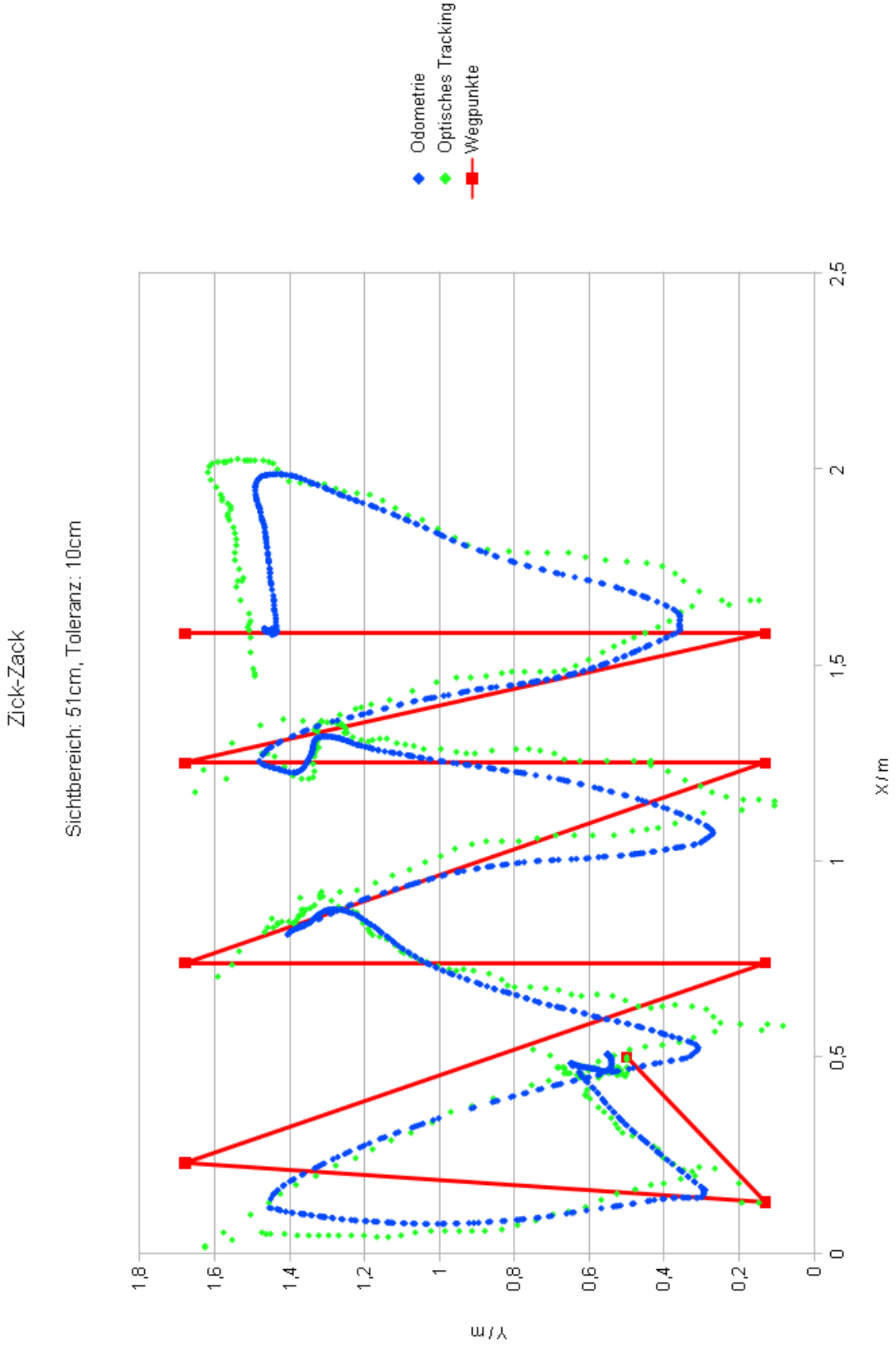

Zick-Zack
Sichtbereich: 21cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
-1


Zick-Zack
Sichtbereich: 30cm, Toleranz: 0cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5


Zick-Zack
Sichtbereich: 30cm, Toleranz: 0cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
-0,5
0
0,5
1
1,5
2
2,5


Zick-Zack
Sichtbereich: 30cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2,5
2
1,5
1
0,5
0
-0,5
2
1,5
1
0,5
0
-0,5


Zick-Zack
Sichtbereich: 30cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
2
1,5
1
0,5
0
-0,5
2,5
2
1,5
1
0,5
0
-0,5


Zick-Zack
Sichtbereich: 51cm, Toleranz: 10cm
Odometrie
Optisches Tracking
Wegpunkte
Y / m
X / m
0
0,2
0,4
0,6
0,8
1
1,2
1,4
1,6
1,8
0
0,5
1
1,5
2
2,5